UNIVERSITÀ DEGLI STUDI DI SALERNO

FACOLTÀ DI SCIENZE
MATEMATICHE, FISICHE E NATURALI

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

Una rete neurale per il trattamento di segnali geofisici e datazione di profili orografici basata su eventi astronomici

Tesi di Laurea Sperimentale

Relatori
Prof. Bruno D'Argenio, Prof. G. Longo

Candidato
Massimo Brescia

ANNO ACCADEMICO 1993/94

.....dedicato ai miei genitori

INTRODUZIONE

Negli ultimi anni, nel settore delle scienze della terra, si e' notevolmente sviluppato l'interesse verso lo studio di fenomeni ciclici, cioe' di eventi geologici connessi a processi che si ripetono periodicamente, con ciclicita' comprese in un ampio intervallo.

Di particolare interesse risulta essere lo studio dei cicli sedimentari compresi tra i 20 e i 400 Ka, (migliaia di anni), legati alle perturbazioni dell'orbita terrestre, (cicli di Milankovitch), [5].

Il problema e' dunque scoprire se il processo di sedimentazione di successioni rocciose sia stato influenzato dai fenomeni ciclici accennati sopra, ed eventualmente, quantificarne il numero, [4], [5].

Successivamente, si dovra' procedere a correlare il periodo temporale di ciascuno di essi con la corrispondente ampiezza, (in termini di centimetri), dello strato sedimentario, [5].

Fino ad oggi il problema e' stato affrontato con metodi tradizionali, in particolare, l'istituto Geomare Sud di Napoli ha proceduto secondo il seguente schema, [5] :

- assimilazione di una rappresentazione della successione rocciosa mediante sequenza di codici numerici, ottenuti attraverso osservazioni, dirette, campionate ad intervalli regolari, (in genere a intervalli di un centimetro), di caratteristiche sedimentologiche della successione, (tessitura, diagenesi, colorazione etc.).

Il risultato di questa fase consisteva in un segnale, (onda quadra), rappresentante la successione stessa.

- analisi spettrale di tale segnale, mediante tecniche classiche basate sull'analisi di Fourier per l'estrapolazione delle armoniche fondamentali di esso.Il risultato era l'ottenimento delle frequenze spaziali significative del

segnale.L'inverso di tali frequenze permetteva di riconoscere gli spessori in centimetri degli strati rappresentanti le periodicita' spaziali presenti nella successione.

- associazione spazio/tempo tra tali periodicita' spaziali e i periodi temporali relativi ai sei cicli principali di Milankovitch corrispondenti ad altrettanti fenomeni di perturbazione dell'orbita terrestre. La difficolta' a questo punto della procedura era dovuta alla necessita' di confrontare quantita' espresse in unita' di misura tra loro completamente differenti, nella fattispecie centimetri/anni, per le quali non esiste un rapporto oggettivo immediato. Tale difficolta' era stata superata rendendo adimensionate le due quantita', mediante la realizzazione di due matrici, una contenente i rapporti tra le periodicita' spaziali e l'altra tra quelle temporali, al fine di ottenere semplici entita' numeriche, facilmente confrontabili,[25].Dal confronto delle due matrici si risaliva cosi' al coefficiente di correlazione che esprimeva la durata in anni del processo di sedimentazione dei vari strati della successione.

Connesso al primo passo della procedura or ora descritta, vi e' un ulteriore problema: a causa di alcuni fenomeni naturali, l'osservazione diretta della successione incontra sovente delle interruzioni, di cui non e' possibile quantificarne la durata in maniera precisa. Per questo motivo i geologi sono costretti a suddividere la sequenza in porzioni piu' o meno lunghe e a considerare piu' attendibile, per un'analisi successiva, quella con la durata maggiore, [22], [23].

Nel presente lavoro di tesi, abbiamo affrontato dunque i seguenti problemi :

a) determinazione dei cicli di Milankovitch presenti nel segnale stratigrafico.
b) ricostruzione delle interruzioni.

Per il punto a) abbiamo seguito un approccio completamente originale, mediante l'utilizzo di un modello di rete neurale, il Percettrone Multistrato, (PMS), con algoritmo di Back - Propagation, (BP), preceduto da una fase di "preprocessing", in cui abbiamo realizzato un'implementazione del metodo di analisi spettrale, proposto da Scargle, [6], basato sul Periodogramma, oltre ad una successiva fase, del tutto personale, di preparazione dei segnali con cui far apprendere la rete neurale.

Per il punto b) e' stato elaborato un modello di predittore ad un passo, basandosi sulla teoria dell'analisi multivariata di sequenze, [31], costituito dalla stessa rete neurale precedentemente descritta, per la predizione di un centimetro della successione, note porzioni precedenti di essa di dimensioni variabili.

Data l'estrema originalita' degli approcci teorizzati ed il carattere visibilmente sperimentale del presente lavoro, nonche' la limitatezza del periodo a disposizione per portare a compimento tali studi, la nostra ricerca e' stata in gran parte concentrata sul punto a), limitando gli sforzi, (temporaneamente), riguardo l'analisi del punto b).

Omettiamo per brevita' l'elencazione di tutti i modelli teorizzati per l'analisi del punto a),[9], [10], [14], [15], che, pur se affascinanti dal punto di vista teorico, hanno fornito scarsi risultati da un punto di vista prettamente sperimentale.Ci riserviamo quindi di approfondire nel prossimo futuro entrambi gli aspetti del problema descritto, nonche' di migliorare la performance dei modelli realizzati.

Sono quindi da intendersi sotto tale aspetto i risultati riportati in queste sede, rispettivamente positivi per il punto a) e mediocri per il punto b).

E' superfluo ricordare che l'ostacolo principale per la ricerca scientifica e' la finitezza del tempo!

1. ANALISI DEI SEGNALI

1.1 SEGNALI A TEMPO DISCRETO

Un segnale si può' definire come una funzione rappresentante una grandezza fisica in termini dei suoi stati e dei suoi comportamenti.L'informazione che possiede un segnale risiede in genere nelle variazioni della variabile tempo e delle tre componenti dello spazio.Dunque,da un punto di vista matematico un segnale si puo' rappresentare nella sua forma più generale come funzione di 4 variabili indipendenti.Ad esempio,inerentemente al nostro caso,una fotografia puo' essere rappresentata come una funzione,in tal caso rappresentante la luminosita',di due variabili spaziali.

Poiche' la variabile indipendente,(tempo), della rappresentazione di un segnale puo' essere discreta o continua ,si parlera' di segnali a tempo discreto o continuo.Oltre alle variabili del sistema,anche i valori del segnale possono essere continui o discreti : i **segnali numerici** sono quelli per cui sia il tempo che l'ampiezza sono discreti; i **segnali analogici** sono quelli per cui sia il tempo che l'ampiezza sono continui.Per i nostri scopi dedicheremo maggiormente la nostra attenzione a quelli numerici.

Come meglio specificheremo in seguito,un segnale puo' essere rappresentato da una sequenza.Una sequenza di numeri x,in cui l'n-esimo numero di essa e' indicato con x(n),e' scritta formalmente come

$$x = \{x(n)\} \quad -\infty < n < +\infty .$$

Denoteremo x(n) come il campione n-esimo della sequenza.E' importante a questo punto ricordare che x(n) e' definito solo per valori interi di n.Un esempio molto importante di sequenza e' la **sequenza campione unitario**, [1], definita come : $\delta(n) = \begin{cases} 1 & n=0 \\ 0 & n\neq 0 \end{cases}$. Tale funzione e' anche denominata impulso a tempo discreto.

Un altro esempio e' la **sequenza gradino unitario** , [1], definita come : $u(n)=\begin{cases}1 & n\geq 0\\ 0 & n<0\end{cases}$. Quest'ultimo e' legato al precedente dalle seguenti equazioni :

$$u(n)=\sum_{k=-\infty}^{n}\delta(k) \quad \text{e} \quad \delta(n)=u(n)-u(n-1).$$

Un'altra molto importante e' la **sequenza esponenziale complessa,** [1]. Essa e' della forma $e^{(\sigma+j\omega)n}$. Tali sequenze sono periodiche con periodo $\frac{2\pi}{\omega}$ se σ = 0 e solo se il periodo e' un numero intero. Se e' invece razionale,saranno ancora periodiche ma con periodo piu' lungo e infine se e' irrazionale,le sequenze non saranno piu' periodiche. Il parametro ω e' la frequenza della sinusoide o esponenziale complesso; esso puo' essere scelto in un insieme continuo di valori.

Introduciamo inoltre l'energia di una sequenza, E = $\sum_{n=-\infty}^{\infty}|x(n)|^2$.

Proprieta' importanti delle sequenze sono :

a) si dice che una sequenza y e' la versione ritardata della sequenza x se y(n) = x(n - m), con m intero.

b) una sequenza arbitraria puo' essere espressa come la somma di campioni unitari ritardati, cioe' :

$x(n)=\sum_{k=-\infty}^{\infty}x(k)\delta(n-k)$.

c) se y(n) e' in relazione con le sequenze x(n) e h(n) secondo una delle seguenti equivalenti formule :

$$y(n)=x(n)*h(n)=\sum_{k=-\infty}^{\infty}x(k)h(n-k)$$
$$y(n)=h(n)*x(n)=\sum_{k=-\infty}^{\infty}h(k)x(n-k)$$

, si dice che y(n) e' la **convoluzione** delle 2 sequenze, [2].

Nella teoria dei sistemi lineari invarianti alla traslazione e' particolarmente utile la rappresentazione dei segnali sottoforma di sinusoidi o di esponenziali complessi.

Infatti in tal caso la risposta di un sistema siffatto ad un ingresso sinusoidale e' ancora sinusoidale con la stessa frequenza dell'ingresso, con ampiezza e fase determinate dal sistema. Scendendo un po' in dettaglio, sia x(n) = $e^{j\bullet n}$; consideriamo la convoluzione con la **risposta al campione unitario h(n)** [1] **:**

$$y(n)=\sum_{k=-\infty}^{\infty}h(k)e^{j\omega(n-k)}=e^{j\omega n}\sum_{k=-\infty}^{\infty}h(k)e^{-j\omega k}$$

Definiamo allora H($e^{j\omega}$) = $\sum_{k=-\infty}^{\infty}h(k)e^{-j\omega k}$ **(1.1)**

Avremo allora che y(n) = $H\left(e^{j\omega}\right)e^{j\omega n}$ **(1.2) .** $H\left(e^{j\omega}\right)$ e' detta **risposta in frequenza del sistema,** [1].

Questa funzione complessa e' continua e periodica di periodo 2π. La **(1.1)** esprime $H\left(e^{j\omega}\right)$ nella forma di una serie di Fourier i cui coefficienti sono i valori della risposta al campione unitario , (FT).

Per cui $h(n)=\frac{1}{2\pi}\int_0^{2\pi}H\left(e^{j\omega}\right)e^{j\omega n}d\omega$ **(1.3).** La **(1.3)** rappresenta la sequenza h(n) come sovrapposizione di segnali esponenziali le cui ampiezze sono date dalla **(1.1)**. La rappresentazione di una sequenza per mezzo della trasformata **(1.1)** si puo' estendere a qualsiasi sequenza purche' la **(1.1)** converga. Si puo' quindi definire per una generica sequenza x(n) la FT come $X\left(e^{j\omega}\right)=\sum_{n=-\infty}^{\infty}x(n)e^{-j\omega n}$ e la trasformata inversa come $x(n)=\frac{1}{2\pi}\int_0^{2\pi}X\left(e^{j\omega}\right)e^{j\omega n}d\omega$.

Facciamo ora un breve cenno alle relazioni fra segnali a tempo discreto con quelli a tempo continuo. In genere i segnali discreti sono derivati da quelli a tempo continuo mediante un campionamento periodico. E' dunque importante capire come la sequenza cosi' ottenuta sia legata al segnale originale.

Sia un segnale analogico $x_a(t)$ avente la seguente rappresentazione di Fourier :

$$x_a(t) = \frac{1}{2\pi}\int_0^{2\pi} X_a(j\omega)e^{j\omega t}\,d\omega$$
$$X_a(j\omega) = \int_{-\infty}^{\infty} x_a(t)e^{-j\omega t}\,dt \qquad \textbf{(1.4)}$$

La sequenza x(n) con valori $x(n) = x_a(nT)$ si dice essere ottenuta da $x_a(t)$ mediante campionamento periodico con T periodo di campionamento. L'inverso di T viene detto frequenza di campionamento. Per vedere in che modo x(n) rappresenta il segnale $x_a(t)$,conviene mettere in relazione la FT continua $X_a(j\omega)$ di $x_a(t)$ con la FT discreta $X(e^{j\omega})$ di $x(\omega)$. La relazione e'

$$X\left(e^{j\omega}\right) = \frac{1}{T}\sum_{r=-\infty}^{\infty} X_a\left(\frac{j\omega}{T} + j\frac{2\pi r}{T}\right) \qquad \textbf{(1.5)} .$$

1.2 ALIASING E TEOREMA DI NYQUIST

Riportiamo un importante risultato :

teorema di campionamento, [30] :

Data una funzione f(t) che abbia uno spettro di ampiezza compreso in un intervallo [0,t],esso risulta completamente determinato quando sono conosciuti i valori di campioni ad istanti t_n intervallati da un tempo di ampiezza $\Delta T = T_o = \frac{1}{2t}$

dove t e' la banda del segnale,ΔT e' l'intervallo di campionamento di Nyquist e t_n = n/2t sono gli istanti di campionamento, per t_n = 0,± 1,± 2,.....

♦

Ora, se $\frac{\omega}{2} < \frac{\pi}{T}$, cioe' se campioniamo ad una frequenza che e' almeno due volte maggiore della piu' alta frequenza della trasformata continua $X_a(j\omega)$ allora $X\left(e^{j\omega}\right)$ e' identica a $X_a\left(\frac{\omega}{T}\right)$ in $-\pi \le \omega \le \pi$.

In tal caso e' ragionevole attendersi che $x_a(t)$ possa essere ricostruita dai campioni $x_a(nT)$ attraverso un'opportuna formula d'interpolazione.

La frequenza di campionamento corrispondente a 2 volte la massima frequenza della trasformata a tempo continuo si chiama **frequenza di Nyquist**, [30].

Dunque, assumendo $\omega/2 < \pi/T$, la formula cercata e' fornita dal fondamentale

teorema di Nyquist, [30] :

Dato un segnale x(t) Fourier-sviluppabile, e' possibile campionarlo senza grossi danni, purche' $K = \frac{1}{T_o} \ge 2B$, ove B e' l'ampiezza di banda del segnale; inoltre esso e' completamente determinato dall'insieme di valori x(KT / N+1) e puo' essere fornito ad ogni istante dalla seguente formula :

$$x(t)=\sum_{k=0}^{N} x\left(\frac{KL}{N+1}\right)\frac{sin\left\{\frac{N+1}{2}\pi\left(t-\frac{KL}{N+1}\right)\right\}}{(N+1)sin\left\{\frac{\pi}{2}\left(t-\frac{KL}{N+1}\right)\right\}}\ 0\le t\le L$$

,ove L e' la durata totale della funzione analogica. Vedi fig.1.

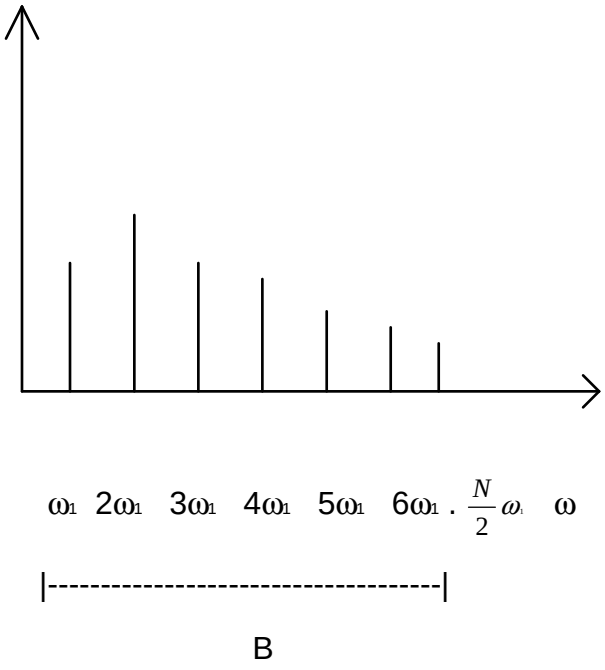


Fig. 1

Se il periodo di campionamento e' troppo lungo, cioe' $\omega/2 > \pi/T$, le repliche traslate di $X_a(j\omega/T)$ si sovrappongono. In tal caso le alte frequenze di $X_a(j\omega)$ si riflettono nelle basse frequenze di $X\left(e^{j\omega}\right)$, cioe' una componente ad alta frequenza si trasforma in una a più' bassa frequenza. Tale fenomeno si chiama **aliasing**, [30].

1.3 SEQUENZE BIDIMENSIONALI

Molti problemi importanti di elaborazione di segnali comportano la trattazione segnali multidimensionali. Una sequenza bidimensionale e' una funzione di 2 variabili intere. Definiamo quindi le sequenze viste prima nel caso bidimensionale. La sequenza campione unitario e' $\delta(m,n)=\begin{cases}1 & m=n=0\\ 0 & m,n\neq 0\end{cases}$ .Mentre la sequenza gradino unitario e' $u(m,n)=\begin{cases}1 & m\geq 0,n\geq 0\\ 0 & \text{else}\end{cases}$. Valgono inoltre le stesse proprieta' di prima:

a) $y(m,n)=x(n-k,m-r)\quad k,r$ interi

b) $x(m,n)=\sum_{k=-\infty}^{\infty}\sum_{r=-\infty}^{\infty}x(k,r)\delta(m-k,n-r)$.

c) $y(m,n)=\sum_{k=-\infty}^{\infty}\sum_{r=-\infty}^{\infty}x(k,r)h(m-k,n-r)$

Inoltre se $x(m,n)=e^{j\omega_1 m}e^{j\omega_2 n}\Rightarrow y(m,n)=\sum_k\sum_r h(k,r)e^{j\omega_1 m}e^{j\omega_2 n}e^{-j\omega_1 k}e^{-j\omega_2 r}=$

$=H(e^{j\omega_1},e^{j\omega_2})e^{j\omega_1 m}e^{j\omega_2 n}$.

Sara' dunque $h(m,n)=\frac{1}{4\pi^2}\int_0^{2\pi}\int_0^{2\pi}H(e^{j\omega_1},e^{j\omega_2})e^{j\omega_1 m}e^{j\omega_2 n}d\omega_1 d\omega_2$.

Inoltre la FT e' $X(e^{j\omega_1},e^{j\omega_2})=\sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty}x(m,n)e^{-j\omega_1 m}e^{-j\omega_2 n}$ e

l'inversa e' $x(m,n)=\frac{1}{4\pi^2}\int_0^{2\pi}\int_0^{2\pi}X(e^{j\omega_1},e^{j\omega_2})e^{j\omega_1 m}e^{j\omega_2 n}d\omega_1 d\omega_2$.

1.4 TRASFORMATA DI FOURIER

Il tipo di segnale che utilizzeremo nel seguito e' definito nel dominio dello spazio, essendo il risultato di osservazioni stratigrafiche di pareti montane.Poiche' i metodi di analisi spettrale adoperati si basano sulla rappresentazione del segnale nel dominio delle frequenze, e' necessario introdurre alcune nozioni elementari sul processo di trasformazione realizzato sui dati spaziali a nostra disposizione.

Nel seguito di questo paragrafo faremo uso della variabile tempo per semplicita' di notazione, ipotizzando che per la nostra discussione valga la relazione 1 cm = 1 s.

Allora, dato un segnale continuo, X(t), la scelta di un tasso di campionamento implica la trasformazione del segnale in serie discreta. E' noto che una tale sequenza puo' avere una rappresentazione nel dominio delle frequenze, misurate in Hz, (1 Hz = 1/s).

Supponiamo dunque di avere la serie discreta x(t), ottenuta dal campionamento di X(t) ad una frequenza di h Hz. Cio' significa che consideriamo i valori del segnale continuo a intervalli 1/h di tempo. Quindi Δt = intervallo di campionamento = 1/h s.

La scelta della frequenza h implica la supposizione che il segnale originale, a priori rumoroso, sia formato da armoniche aventi frequenze $\leq$ h, altrimenti non sarebbe possibile evidenziarle a posteriori. Quindi definendo h, ipotizziamo che il segnale contenga frequenze comprese nella banda [0, h[.

Supponendo che il numero di punti costituenti la serie x(t) sia N, calcoliamo allora la **trasformata di Fourier discreta**, [2], di essa mediante la seguente formula : $y = ft(x, N) = y(k) = \sum_{t=0}^{N-1} x(t)\, e^{-\frac{2\pi i k t}{N}}$, $k = 0, \ldots, N-1$

Le y(k) rappresentano i valori d'ordinata nello spettro di Fourier.

Per la scelta dei valori d'ascissa dello spettro si scelgono allora tutte le frequenze comprese nella banda [0, h/2[, (ci fermiamo a h/2 poiche' i valori dello spettro per la restante meta' dell'intervallo di ascissa sono simmetrici a quelli precedenti), utilizzando la formula $f = h * \frac{k/2}{N}$, $k = 0,...,N-1$, $f \in [0, h/2[$

Otteniamo cosi' uno spettro di Fourier che puo' essere analizzato con i vari metodi di analisi spettrale.Ovviamente la scelta della frequenza o velocita' di campionamento Vc, (Vc = h Hz), deve rispettare il vincolo imposto dal teorema di Nyquist, secondo cui : detta Fny = Δt/2 la frequenza di Nyquist, Vc $\geq$ 2 Fny.

In tal modo si e' sicuri di campionare il segnale evitando possibili fenomeni di aliasing, o sovrapposizione di armoniche, [22].

1.5 ANALISI DEI SEGNALI DISCRETI

Nei paragrafi precedenti abbiamo visto come i segnali a tempo discreto hanno una rappresentazione sia nel dominio del tempo sia in quello della frequenza. Finora abbiamo assunto che i segnali siano deterministici, cioe' ogni valore di una sequenza e' univocamente determinato o da un'espressione matematica o da una tabella di dati. Per i segnali periodici abbiamo visto che, essendo per definizione identici da periodo a periodo, possono essere rappresentati univocamente in termini di un singolo periodo. Ma un singolo periodo e' una sequenza di durata finita e quindi anche di energia finita. Quest'ultima proprieta' e' stata usata per darne una rappresentazione in serie e trasformata di Fourier discreta.

In realta' esistono molti importanti tipi di segnali che o non hanno energia finita o non sono periodici . In numerose situazioni, come la nostra, i processi che danno origine ai segnali sono cosi' complessi da rendere molto difficile la descrizione precisa di un segnale [2]. Molti degli effetti di imprecisione, sulla ricostruzione del segnale, si possono rappresentare mediante rumore additivo, per il quale una descrizione piu' conveniente e' una sequenza a energia infinita.

La chiave per la rappresentazione matematica di tali tipi di segnali sta nella loro descrizione in termini di medie [26]. Molte proprieta' di questi segnali si possono esprimere sinteticamente per mezzo di due sequenze a energia finita,chiamate **sequenza di autocorrelazione e sequenza di autocovarianza [1]**, per le quali esistono le rispettive FT. Infatti, come vedremo, le loro FT hanno un'interessante interpretazione in termini di distribuzione in frequenza della potenza del segnale.

Nello sviluppare la teoria matematica dei segnali a energia infinita, e' conveniente lavorare nell'ambito dei segnali non deterministici. Da questo punto di vista il segnale e' considerato come membro di un insieme di segnali a tempo

discreto caratterizzato da un insieme di funzioni di densita' di probabilita' [26]. Il concetto fondamentale nella rappresentazione matematica dei segnali a energia infinita e' quello di processo casuale, (p. c.).

Data una sequenza di durata arbitraria, l'n-esimo valore di essa, x(n), puo' essere pensato come una variabile aleatoria, e l'insieme di esse {x(n)} e' una sequenza campione del processo casuale. Da un punto di vista dell'elaborazione numerica dei segnali, una particolare sequenza di dati si puo' interpretare come un membro dell'insieme delle sequenze campione di un processo casuale. Il maggiore problema e' che, dato un segnale a tempo discreto appartenente ad un insieme di sequenze campione, la legge di probabilita', ossia la struttura del corrispondente processo casuale, che gli e' associata non e' generalmente nota [1].In molte applicazioni i processi casuali servono come modelli dei segnali, nel senso che un particolare segnale a energia infinita puo' essere considerato una sequenza campione di un processo casuale.

Alcune proprieta' medie dell'insieme possono tuttavia essere previste se e' nota la legge di probabilita' del processo.

Poiche' un processo casuale e' un insieme ordinato di variabili casuali, si puo' caratterizzare il processo mediante medie statistiche delle variabili che lo costituiscono; allora, accanto alla media, la media quadratica e la varianza di un processo casuale, che forniscono solo una piccola quantita' di informazione sul processo, altre medie piu' utili sono:

la **sequenza di autocorrelazione [1]** che e' definita come: $\Phi_{xx}(n,m) = E\left[x_n, x_m^*\right] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x_n x_m^* p_{x_n x_m}(x_n, n, x_m, m)\,dx_n dx_m$, dove lo $*$ indica il complesso coniugato.

La **sequenza di autocovarianza [1]** di un processo casuale che e' definita come: $\gamma_{xx}(n,m) = E\left[(x_n - m_{x_n})(x_m - m_{x_m})^*\right]$.

La relazione tra le due e' la seguente: $\gamma_{xx}(n,m) = \Phi_{xx}(n,m) - m_{x_n} * m_{x_m}$

L'autocorrelazione e' una misura della dipendenza fra i valori di un processo casuale in tempi diversi. Essa descrive la variazione nel tempo di un segnale casuale. Viceversa la sequenza di correlazione e la funzione di covarianza si possono usare per misurare la dipendenza tra due diversi processi casuali e sono indicate rispettivamente come $\Phi_{xy}(n,m)$ e $\gamma_{xy}(n,m)$ se x_n e x_m sono due processi casuali. In processi stazionari, mentre le medie del primo ordine sono indipendenti dal tempo, le medie del secondo ordine come l'autocorrelazione sono dipendenti dalla differenza temporale di due istanti diversi.Come abbiamo accennato la nozione di insieme di segnali ad energia infinita e' un concetto utile perche' ci consente di usare la teoria della probabilita' per rappresentare segnali ad energia infinita, tuttavia, sarebbe preferibile avere a che fare con una singola sequenza piuttosto che con un insieme infinito di esse e che la media aritmetica di un gran numero di campioni di una singola sequenza approssimasse il valor medio del processo [26]. Per formalizzare cio' definiamo la media temporale di un p. c. come $< x_n >= \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} x_n$.

Analogamente, la sequenza di autocorrelazione nel tempo e' definita come: $< x_n, x_{n+m} >= \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} x_n x_{n+m}^*$. Si puo' dimostrare, [1], che i limiti ora definiti esistono se x_n e' un processo stazionario con media finita. In generale queste due medie sono funzione di un insieme infinito di variabili casuali e quindi sono esse stesse delle variabili casuali. Tuttavia in condizioni di ergodicita' delle sequenze, le medie temporali di tutte le possibili sequenze campione sono identiche. Cio' significa che per ogni singola sequenza campione {x(n)} per $-\infty<n<+\infty$, si ha :

$$< x(n) >= \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} x(n) = E[x_n] = m_x$$

e

$$< x(n)x^*(n+m) >= \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} x(n)x^*(n+m) = E[x_n, x^*_{n+m}] = \Phi_{xx}(m).$$

In generale un processo casuale per cui le medie temporali sono uguali alle medie del primo ordine si chiama **processo ergodico**. Sebbene per un segnale ad energia infinita la FT non esiste, le sue sequenze di autocorrelazione e di autocovarianza, sono sequenze aperiodiche, per le quali la FT esiste. Quindi come vedremo la rappresentazione spettrale gioca un ruolo importante nell'analisi dei segnali shift-invarianti ad energia infinita.

1.6 SPETTRO DI POTENZA

Definiamo lo spettro di potenza [2], $P_{xx}(\omega)$, come la FT della sequenza di autocorrelazione o di autocovarianza. Cio' comporta qualche difficolta' quando m_x e' diversa da zero perche' $\Phi_{xx}(m)$ tende a m_x^2 quando $m \to \infty$;

infatti, la FT di $\Phi_{xx}(m)$ non esiste se m_x e' diverso da zero, a meno che non ammettiamo la presenza di un impulso nello spettro di potenza per $\omega = 0$. Osserviamo che quando m_x e' = 0, le sequenze di autoc. e di autocov. sono identiche e tali sono anche le loro FT. $P_{xx}(\omega)$ e' una funzione simmetrica, cioe' : $P_{xx}(\omega) = P_{xx}(-\omega)$, ed inoltre e' non negativa. Quando una sequenza da elaborare deriva da un campionamento di un segnale analogico a banda limitata, la limitazione dovuta alla lunghezza finita comporta che nel processo di conversione da analogico a numerico siano disponibili soltanto un numero finito di possibili valori per ogni campione. Vedremo che questo effetto puo' essere valutato in termini di rumore aggiuntivo.

1.7 STIMA DELLO SPETTRO DI POTENZA

Uno dei metodi applicativi di rilievo per l'elaborazione numerica dei segnali ed in particolare per la FFT e' quello della stima delle funzioni di autocov. e di spettro di potenza di una sequenza casuale, in particolare in presenza di rumore [1]. Quando si caratterizza un segnale come processo aleatorio occorre spesso stimare delle medie del processo a partire da una singola sequenza campione del processo stesso, cioe' una sequenza x(n) assunta essere una realizzazione di un processo casuale definito dall'insieme delle variabili aleatorie {x(n)}. Inoltre per poter calcolare le stime occorre ricavarle dal segmento finito della sequenza campione x(n). Questo e' soprattutto vero nel caso dei processi ergodici.

Consideriamo ad esempio un processo aleatorio stazionario {x(n)} con $-\infty<n<+\infty$. Il suo valore medio m_x e' definito come, [1],

$m_x = E[x(n)] \int_{-\infty}^{+\infty} x p_x(x) dx \quad \forall m$ e la sua media temporale e' invece :

$m_x =< x(n) >= \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} x_n$. Pertanto la media temporale di ogni sequenza campione e' uguale a : $< x(n) >= \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} x(n)$. Supponiamo che essa sia uguale ad m_x. La varianza del processo aleatorio e' definita come:

$\sigma_x^2 = E\left[(x_n - m_x)^2\right] =< (x_n - m_x)^2 >$.

La sequenza di autocov. e' definita come:

$\gamma_{xx}(m) = E\left[(x_n - m_x)(x^*_{n+m} - m^*_x)\right] =< (x_n - m_x)(x^*_{m+n} - m^*_x) >$, e la densita' spettrale di potenza come : $P_{xx}(\omega) = \sum_{m=-\infty}^{\infty} \gamma_{xx}(m) e^{-j\omega m}$. Per stimare un parametro α di un processo aleatorio si ha in generale a disposizione un segmento finito di una singola sequenza campione, cioe' in generale N valori x(n). La stima $\hat{\alpha}$ del parametro α e' dunque funzione della v.a. x_n. cioe' : $\hat{\alpha} = F(x_0, x_1, ..., x_{n-1})$, e quindi

anch'essa e' una v.a.. Indichiamo con $P_{\hat{\alpha}}\left(\hat{\alpha}\right)$, la funzione di densita' di probabilita' di $\hat{\alpha}$, (f. d. p.). La sua forma e il suo andamento dipenderanno dalla scelta dello stimatore F e dalla densita' di probabilita' delle variabili x(n). Definiremo uno stimatore buono se e' elevata la probabilita' che la stima sia prossima ad α [1]. In generale ha senso dire che per uno stimatore buono la funzione densita' di probabilita' deve essere stretta e concentrata intorno ad α. Le proprieta' degli stimatori che sono usate come base per il confronto sono la polarizzazione e la varianza.

La polarizzazione di uno stimatore e' definita come il valore vero del parametro α - il valore atteso della stima, ovvero si ha che :

$\text{polarizzazione} = \alpha - \mathrm{E}\left[\hat{\alpha}\right] \triangleq \mathrm{B}$. Uno stimatore e' non polarizzato se B e' nulla, cio' significa che il valore atteso della stima e' il valore vero α. Quindi se la f.d.p. e' simmetrica il suo centro cade esattamente sul valore α, [1].

La varianza dello stimatore e' una misura della larghezza della densita' di probabilita' ed e' definita da: $\mathrm{var}\left(\hat{\alpha}\right) = E\left[\left(\alpha - \mathrm{E}\left[\hat{\alpha}\right]\right)^2\right]$. Una varianza piccola implica che la f.d.p. e' concentrata intorno al suo valore medio. In molti casi il confronto fra due stimatori e' complicato dal fatto che quello con la polarizzazione minore ha la varianza maggiore o viceversa, per cui e' opportuno considerare l'errore quadratico medio associato ad uno stimatore definito come: $E\left[\left(\hat{\alpha} - \alpha\right)^2\right] = \mathrm{var}\left(\hat{\alpha}\right) - B^2$. Uno stimatore si dice consistente se la polarizzazione e la varianza tendono entrambi a zero al crescere del numero di osservazioni, [1].

1.8 PERIODOGRAMMA

In base alle considerazioni fatte saremmo tentati di concludere che le FT di queste stime della sequenza di autocov. forniscono buone stime dello spettro di potenza. Sfortunatamente, cio' non e' vero, poiche' la varianza in generale non tende a zero al crescere della lunghezza della sequenza, pero' vedremo che e' possibile ricavare una buona stima dello spettro di potenza smussando la FT della stima della covarianza. Consideriamo il caso in cui si prenda come stima dello spettro di potenza la FT della stima polarizzata dell'autocorrelazione $C_{xx}(m)$, definita da : $C_{xx}(m) = \frac{1}{N} \sum_{n=0}^{N-m-1} x(n)x(n+m)$. La polarizzazione di C segue dal fatto

che : $E[C_{xx}(m)] = \left(\frac{N-|m|}{N} \right) \Phi_{xx}(m)$.

In particolare $C_{xx}(m)$ vale : $pol = \Phi_{xx}(m)\frac{m}{N}$ .La FT di $C_{xx}(m)$ e' $I_N(\omega) = \frac{1}{N}\left|x\left(e^{j\omega}\right)\right|^2$; questa e' la def. di **periodogramma, (P.), [1]**.

Come nei casi precedenti e' interessante trovare la polarizzazione e la varianza del periodogramma. Calcoliamo intanto la media del P. :

$$E[I_n(\omega)] = \sum_{m=-(N-1)}^{N-1} E[C_{xx}(m)]e^{-j\omega m} .$$

D'altro canto dalla definizione della media di $C_{xx}(m)$ si ha che: $E[I_n(\omega)] = \sum_{m=-(N-1)}^{N-1} \frac{N-|m|}{N} \Phi_{xx}(m)e^{-j\omega m}$. Dal fatto che la sommatoria e' finita e per la presenza della frazione costante, la media non coincide con la FT della sequenza di autocorrelazione Φ_{xx}, pertanto il periodogramma risulta una stima polarizzata dello spettro di potenza. La formula della media si puo' interpretare come FT della sequenza di autocorrelazione pesata con la seguente finestra :

$w_B(m) = \begin{cases} \frac{N-|m|}{N} & se\ |m|<N \\ 0 & \text{else} \end{cases}$. Quindi la media del P. e' interpretabile nel dominio della frequenza come la convoluzione

$$E[I_n(\omega)] = \frac{1}{2\pi}\int_{-\pi}^{\pi} P_{xx}(\theta) w_B\left(e^{j(\omega-\theta)}\right) d\theta \text{ dove } w_B\left(e^{j\omega}\right) = \frac{1}{N}\left(\frac{\sin\left(\omega\frac{N}{2}\right)}{\sin\left(\frac{\omega}{2}\right)}\right)^2 .$$

Per ottenere la varianza del P. ipotizziamo che la sequenza x(n) sia una sequenza campione di un processo reale bianco e gaussiano.

$$I_N(\omega) = \frac{1}{N}\left|X\left(e^{j\omega}\right)\right|^2 = \frac{1}{N}\sum_{l=0}^{N-1}\sum_{m=0}^{N-1} X(l)X(m)e^{j\omega m}e^{-j\omega l} .$$

$$\text{La covarianza del P. e': } \operatorname{cov}(I_N(\omega_1), I_N(\omega_2)) = E[I_N(\omega_1)I_N(\omega_2)] - E[I_N(\omega_1)]E[I_N(\omega_2)]$$

Allora :

$$E[I_N(\omega_1)] = E[I_N(\omega_2)] = \sigma_x^2 \Rightarrow$$

$$\Rightarrow \operatorname{cov}(I_N(\omega_1), I_N(\omega_2)) = \sigma_x^4\left\{\left(\frac{\sin\left[(\omega_1+\omega_2)\frac{N}{2}\right]}{N\sin[(\omega_1+\omega_2)/2]}\right)^2 + \left(\frac{\sin\left[(\omega_1-\omega_2)\frac{N}{2}\right]}{N\sin[(\omega_1-\omega_2)/2]}\right)^2\right\}$$

Dunque, la varianza ad una frequenza ω e'

$\operatorname{var}(I_N(\omega)) = \operatorname{cov}(I_N(\omega), I_N(\omega)) = \sigma_x^4\left\{1+\left(\frac{\sin(\omega N)}{N\sin(\omega)}\right)^2\right\}$. La varianza non tende a zero per N tendente ad infinito, percio' il P. non e' una stima consistente.

Una caratteristica importante del P. e' che pur non essendo uno stimatore consistente, poiche' la varianza non tende a zero, al crescere del numero delle osservazioni, la covarianza tende a zero, quindi i valori del P. al crescere di N diventano scorrelati tra loro ed il P. presenta un numero di oscillazioni crescente.Quando lo spettro non contiene rumore bianco, pur essendo gaussiano, l'analisi e' notevolmente piu' difficile; in tal caso si effettua il calcolo della covarianza tra campioni dello spettro in maniera euristica, [2].

1.9 STUDI SU ASPETTI STATISTICI DELL'ANALISI SPETTRALE.

Il seguente paragrafo analizza l'efficienza di controllo di un segnale periodico nascosto da rumore con uno dei metodi piu' comunemente usati : il Periodogramma, [6]. I campioni a disposizione sono dati spaziati ad intervalli irregolari,[4], [5]. Il segnale qui considerato e' strettamente periodico. Il problema fondamentale e': una variabile fisica X e' misurata ad un insieme di tempi t_i ; le serie temporali dei dati risultanti {X(ti), i=0...n-1}, sono assunte essere la somma di un segnale e di errori di osservazione random : Xi = X(ti) = Xs(ti) + R(ti) **1.9.1**.Inoltre assumeremo che gli errori in tempi differenti siano indipendenti e distribuiti normalmente con media 0 e varianza $\bullet^2$.Cio' che vogliamo e' stabilire l'esistenza di un segnale nonostante la presenza di rumore, effettuare la stima del contenuto armonico e del periodo del segnale.

Una prima operazione consigliata e' alterare la definizione classica del P. in modo che diventi shift-invariante e che risulti piu' semplificata l'analisi statistica. Dunque riportiamo entrambe le versioni del P. :

classica : $P_x(\omega) = \frac{1}{N_0}\left|FT_x(\omega)\right|^2 = \frac{1}{N_0}\left[\left(\sum_j X_j \cos\omega t_j\right)^2 + \left(\sum_j X_j sin\omega t_j\right)^2\right]$.

alterata : $P_x(\omega) = \frac{1}{2}\left\{\frac{\left[\sum_j X_j \cos\omega(t_j-\tau)\right]^2}{\sum_j \cos^2\omega(t_j-\tau)} + \frac{\left[\sum_j X_j sin\omega(t_j-\tau)\right]^2}{\sum_j sin^2\omega(t_j-\tau)}\right\}$,

ove τ e' definita da $\tan(2\omega\tau) = \frac{\left(\sum_j sin2\omega t_j\right)}{\sum_j \cos 2\omega t_j}$.

In ogni caso, il motivo di usare il P. e' che se X contiene una componente sinusoidale di frequenza ω, allora in corrispondenza di tale frequenza i due fattori nella sommatoria $X(t_j)\,\mathrm{ed}\,\mathrm{e}^{-i\omega t}$ risultano in fase fra di loro e danno un grande

contributo alla somma. Mentre per altri valori di ω i termini della somma sono casualmente positivi e negativi e la risultante semplificazione di tali valori genera un piccolo valore della somma, poiche' molti termini si annullano. Per cui la presenza di una sinusoide e' indicata da un grande valore del P. in corrispondenza della frequenza ω, cioe' da uno stretto, limitato picco nello spettro.

Definiamo uno strumento utile : il **tasso segnale/rumore, [6], [7], [8],** definito da : $T = \frac{P_x}{P_R} = \frac{N_0\left(\frac{X_0}{2}\right)^2}{\sigma^2}$, ove P_x e' la potenza del segnale e P_R quella del rumore. Questo strumento mette in luce che per estrapolare la presenza del segnale e' necessario aumentare la potenza del segnale, dato che la varianza del rumore non puo' essere ridotta.Un altro problema e' legato alla scelta della banda di frequenza in cui campionare il segnale cercando di evitare perdita di informazione e l'aliasing.

Poiche' dunque non e' possibile eliminare direttamente il rumore, si cerca di conoscere le frequenze di esso rispondendo alla domanda : qual'e' la probabilita' che una caratteristica spettrale del P. sia dovuta alla presenza del segnale?. La risposta e' contenuta nella conoscenza della distribuzione di probabilita' della variabile aleatoria P(ω), che non e' nota a priori, ma puo' essere ricavata dall'analisi delle medie del P..

2. ANALISI SPETTRALE BASATA SUL PERIODOGRAMMA

Lo studio che vogliamo effettuare si basa sull'analisi dell'organizzazione verticale dei litostrati, successioni di piattaforme segmentarie caratterizzate da successivi affioramenti alternati a sommersioni, [4], [5] . Ci basiamo anche sull'analisi numerica delle caratteristiche sedimentologiche delle successioni rocciose .

Riassumendo quindi, l'analisi spettrale e' una procedura dalla quale si puo' accertare se un'eventuale ricorrenza di sistematiche ripetizioni dei caratteri sedimentologici quali tessiture, strutture diagenetiche, colore e strutture deposizionali, rappresenti o meno un segnale ciclico ed in caso affermativo e' possibile rintracciare le frequenze caratteristiche .In generale, l'analisi spettrale consiste di tre passi fondamentali :

1) Campionamento periodico

2) Prettrattamento del segnale

3) Trattamento del segnale

2.1 CAMPIONAMENTO PERIODICO

Una successione stratigrafica ciclica e' caratterizzata da una ripetizione di determinate caratteristiche quali : litologia, (tipo di roccia), tessitura, colore, spessore etc.....

Questi parametri opportunamente classificati, consentono di ricostruire un record stratigrafico; ogni singolo valore di tale sequenza corrisponde ad una particolare tessitura, individuata mediante osservazione diretta con lente d'ingrandimento o microscopio. In particolare la sequenza del Tobenna, analizzata in questa sede si compone di 9 tipi di tessitura, codificati con valori tra 0.0 e 0.8, [22].

Per analizzare numericamente un segnale stratigrafico, e' necessario immagazzinare la successione estratta . Per fare questo si procede assegnando un periodo o intervallo di campionamento T a cui associare una frequenza di campionamento v = 1/T .Gli intervalli di campionamento da noi usati sono in scala centimetrica o millimetrica, [4], [5] .Cio' corrisponde ad una registrazione centimetro per centimetro della distribuzione dei parametri di cui sopra .

Una condizione, facilmente riscontrabile, che garantisce l'affidabilita' delle registrazioni eseguite e' la "Diagenesi", un processo di cementazione tra i granuli della roccia e la "matrice", cioe' il materiale che riempie i vuoti tra i granuli. Dipende in gran parte dalla posizione della successione rispetto al livello del mare. Nel caso specifico, poiche' la serie carbonatica del Tobenna e' di mare basso, in essa si e' sicuramente verificato il fenomeno di Diagenesi precoce, cioe' un tipo di Diagenesi di relativamente breve durata, [22].

Si ottiene cosi' una rappresentazione del segnale originale attraverso una sequenza di punti discreti .Nella procedura di campionamento e' necessario che l'intervallo T rispetti la condizione data dal teorema di Nyquist :

$v \geq 2f_{Ny}$ dove $f_{Ny} = T/2$.

2.2 PRETRATTAMENTO DEL SEGNALE

Questa fase e' necessaria per eliminare problemi classici dell'analisi spettrale quali "**aliasing"** e "**spectral leakage", (perdita spettrale)** .

Il primo e' responsabile delle sovrapposizioni spurie nello spettro ed e' causato dalla presenza di frequenze al di fuori della banda di Nyquist .

L'altro e' responsabile della comparsa all'interno dello spettro di armoniche spurie, cioe' non appartenenti al segnale originale, ed e' causato dal fatto che il segnale analizzato e' limitato, essendo costituito da un campione finito estratto da un'ipotetico segnale infinito .

Per eliminare il primo, oltre a rispettare il vincolo di Nyquist, si procede ampliando il numero di punti costituenti il segnale originale, ad esempio aggiungendo degli zeri, [5].

Per eliminare il secondo si applica una tecnica di "smoothing" o finestraggio dei dati input, attraverso la loro convoluzione con un'appropriata funzione finestra .Un esempio e' la finestra gaussiana [5], caratterizzata dalla presenza di un lobo centrale e dall'assenza di lobi laterali . Inoltre si applica anche una tecnica di eliminazione dei "trend", cioe' l'eliminazione delle suddette armoniche spurie [22].

Piu' in dettaglio, e' noto che le osservazioni basate su sequenze di dati discrete consistono di due parti : una componente significativa costituente il segnale, ed una, costituente una variazione "random" delle osservazioni detta "noise" o rumore.Quest'ultimo e' un fattore generalmente a breve termine, che fluttua rapidamente .Il segnale, viceversa, tende ad essere a lungo termine .

In altre parole, successivi valori del segnale sono di solito autocorrelati, mentre il rumore in un particolare punto e' completamente indipendente da quello relativo a punti adiacenti ad esso. Per cui, poiche' il segnale presenta una tendenza di fondo, diversamente dal rumore, una media di diversi punti adiacenti tendera' a convergere al valore di tendenza del segnale . Quindi se indichiamo con

Y_j il valore del j-esimo termine della sequenza X , avremo che Y_j sara' formato da un segnale Y_j ed un rumore random ε_j . Il valore del segnale Y_j nel punto x_j puo' essere stimato da Ψ, una media di una serie di osservazioni adiacenti della sequenza . Lo schema da noi utilizzato per tale calcolo e' il seguente [22] :

la sequenza Ψ forma una curva piu' ondulata rispetto a quella rappresentante la sequenza originale, di conseguenza la tecnica prende il nome di "filtraggio dei dati" .Nella fattispecie il nostro schema si basa sulla **media mobile** :

$$\Psi_i = \frac{\sum_{j=i-k}^{i+k} Y_j}{m} \qquad k = m - 1/2 \qquad (2.1)$$

ove m e' la lunghezza dell'intervallo di "smoothing", cioe' il numero di punti su cui e' calcolata la media . L'equazione (2.1) definisce un intervallo centrato attorno al punto da stimare . Notiamo che m deve essere un numero dispari perche' il valore stimato Ψ_i corrisponda al punto centrale . Se m e' pari, sara' stimato un insieme di valori che sono a meta' tra osservazioni adiacenti . Notiamo anche che, poiche' l'intervallo di "smoothing" si estende su m-1/2 osservazioni su entrambi i lati del punto stimato, le osservazioni in prossimita' dei punti iniziali e finali della sequenza non possono essere stimati.

2.3 TRATTAMENTO DEL SEGNALE

A questo punto il segnale spaziale e' pronto per essere trattato. Il primo metodo che analizziamo e' il periodogramma .

Il metodo da noi implementato e' stato suggerito da J.D. Scargle . Esso consiste in una versione modificata del periodogramma classico resasi necessaria dalle seguenti motivazioni [6] :

- i dati provenienti dal campionamento stratigrafico costituiscono una serie non equamente spaziata a causa della presenza di interruzioni, dovute a fenomeni naturali, tra gli strati [4].

- Poiche' e' necessario stimare lo spettro del periodogramma, essendo questo uno stimatore inconsistente per sua natura, si e' resa necessaria l'introduzione di un criterio di selezione delle armoniche del segnale .

Dunque, in alternativa al periodogramma classico, utilizziamo la formula alterata descritta nel paragrafo 1.9 . Scargle propone un valore di soglia determinato da una formula che e' funzione del numero di frequenze e di un parametro di errore fissato [6] :

$$Z_0 = -\log\left(1-\left(1-p_0\right)^{1/N_i}\right) \qquad (2.2)$$

dove : p_0 e' il parametro di errore fissato compreso tra 0 ed 1 . N_i e' il numero di frequenze indipendenti pari ad $N_0/2$ N_0 e' il numero di campioni della sequenza. Questa tecnica di analisi spettrale produce uno spettro in cui compare una serie di periodicita', espresse sotto forma di lunghezze d'onda. Questi valori rappresentano geologicamente gli spessori dei cicli principali costituenti la successione degli strati, ovvero il numero di millimetri o centimetri necessari per la ricomparsa di particolari parametri codificati durante la prima fase . Un'ulteriore fase dell'analisi permette di trasformare gli spessori dei cicli in anni [22], [25].

2.4 ESEMPIO DI APPLICAZIONE E RELATIVI RISULTATI SPERIMENTALI.

L'algoritmo di Scargle e' stato da noi utilizzato su una serie di dati, gentilmente fornitaci dall'istituto Geomare di Napoli, relativa alle osservazioni stratigrafiche del Monte Tobenna, [5], [22], [23], (Salerno , Italia).

La serie e' formata da 5536 valori, campionati con scala centimetrica. Per il nostro esperimento abbiamo realizzato l'implementazione dell'algoritmo di Scargle, aggiungendo l'utilizzo di una sottoprocedura di eliminazione dei "trend", mediante la media mobile e includendo la possibilita' di scelta di una funzione di "smoothing" per il finestraggio. I parametri utilizzati sono stati:

funzione "smoothing" di Hanning [1]: $w(n) = \frac{1}{2}\left[1 - \cos\left(\frac{2n\pi}{N-1}\right)\right]$ $0 \leq n \leq N-1$

valore soglia Zo = 12.524947, con parametro po = 0.01

intervallo di media mobile m = 7, da cui il numero di punti analizzati e'

No = 5539.

numero frequenze indipendenti N = 1300, comprese nella banda [0 , 1/2[

percentuale di finestraggio = 30%.

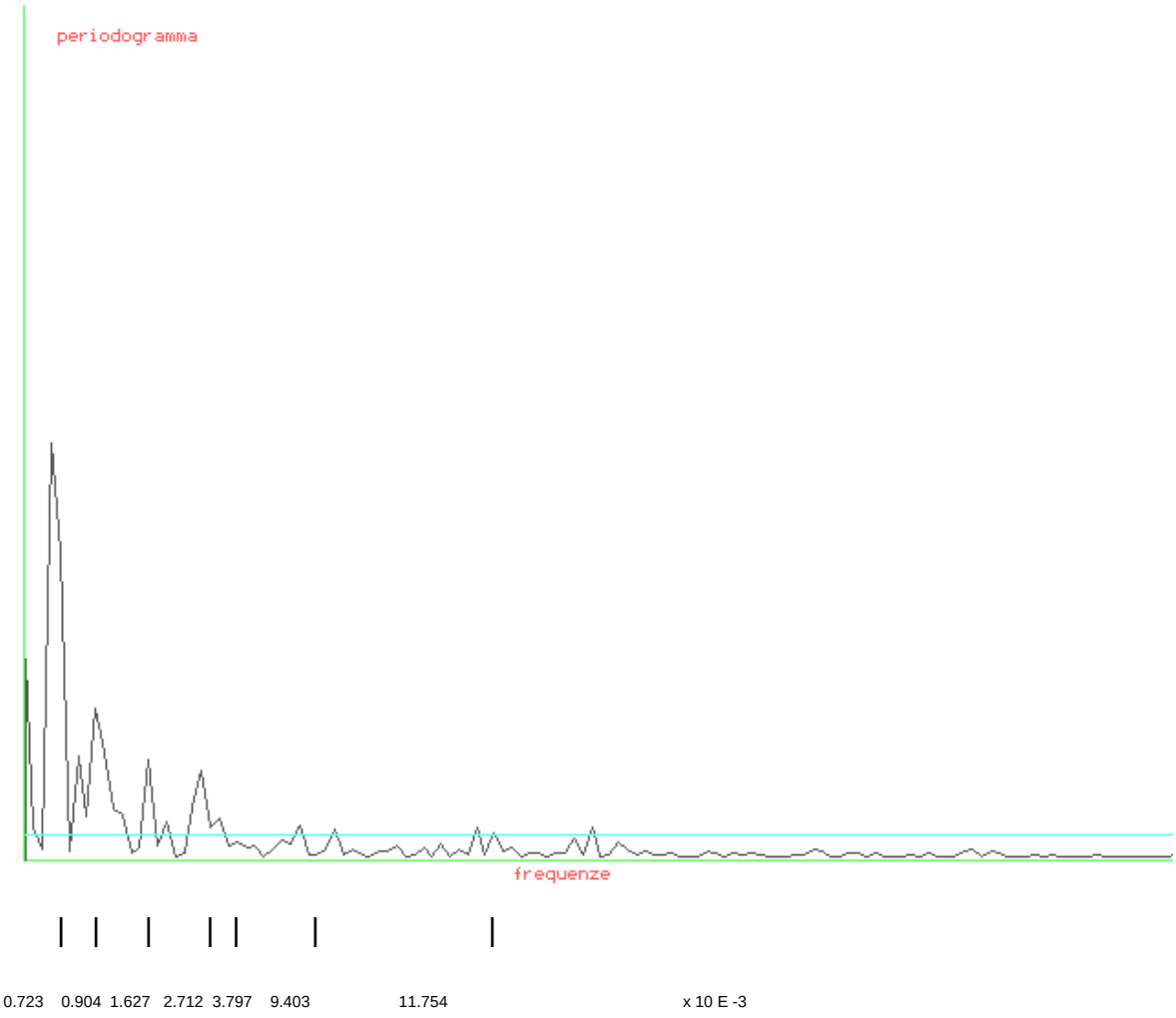
Le frequenze significative estratte dall'algoritmo sono :

frequenze	**ordinata**	**centimetri**
0.000723	226.233459	1383
0.000904	167.966995	1106
0.001627	81.498505	614
0.002712	53.961918	368
0.003797	47.908936	263
0.009403	16.732931	106
0.011754	16.602814	85

I centimetri associati alle frequenze estratte si riferiscono alle ciclicita' piu' significative presenti nel segnale originale, [5].

Una fase successiva che specificheremo in seguito, consiste nell'associare tali ciclicita' spaziali a periodicita' temporali determinate dai cicli di Milankovitch, [5], [22], che presiedono in larga misura alla formazione degli strati geologici.

Riportiamo anche il grafico del periodogramma relativo all'esperimento di cui sopra. Il codice del programma e' descritto in appendice D.



3. IL PROBLEMA DELLA RICOSTRUZIONE DELLE SEQUENZE MANCANTI

Sin qui abbiamo analizzato la possibilita' di evidenziare alcune caratteristiche significative delle sequenze stratigrafiche mediante tecniche standard di analisi spettrale basate su stime della trasformata di Fourier dei dati spaziali. Vi e' pero' un problema di elevata rilevanza legato all'acquisizione di tali sequenze, [22], [23], [25]. Per molteplici cause naturali spesso vi sono delle interruzioni piu' o meno lunghe nella sequenza stratigrafica, che sono causa di perdita d'informazione piu' o meno grave riguardo alle caratteristiche ricercate nel segnale, considerato nella sua interezza. Per tale motivo si rende necessaria la ricerca di tecniche specifiche per avere una ricostruzione il piu' possibile fedele ed attendibile delle parti mancanti del segnale. Per fare cio', occorre innanzitutto identificare un modello descrittivo del sistema in esame, [13], [24].

La difficolta' insita nel problema consiste nel fatto che, come in generale avviene per la descrizione dei fenomeni legati alla scienza della terra, il modello che schematizza il nostro sistema e' di tipo non lineare, la cui trattazione e' dunque di gran lunga piu' difficile. Infatti non sono tuttora disponibili tecniche universalmente adatte alla risoluzione di modelli di sistemi non lineari, per cui si rendono necessari metodi euristici basati su ipotesi testate attraverso tentativi sperimentali.

Sotto questo aspetto le reti neurali, fra cui in particolare il PMS, (Percettrone MultiStrato), possono essere utilizzate con successo per la modellizzazione dei sistemi non lineari basati su misure ingresso - uscita, [13].

3.1 IL MODELLO TEORICO

Volendo dare quindi una rappresentazione formale del modello, consideriamo la rappresentazione NARMAX che ha le seguente espressione, [13],:

y(c) = f(y(c-1),....,y(c-n))

ove,y e' il vettore delle uscite del nostro sistema, cioe' i codici associati agli strati della sequenza.

c e' l'indice della sequenza (in centimetri o millimetri).

f(·) e' una funzione non lineare, non nota a priori.

L'analoga rappresentazione grafica del sistema e' la seguente :

$$\begin{matrix} y(c-1)\rightarrow \\ \vdots \\ y(c-n)\rightarrow \end{matrix} \qquad \mathrm{f}(\bullet) \qquad \Rightarrow \qquad y(c)\cdot$$

Osserviamo due cose :

primo, il codice y(c) del c-esimo centimetro della sequenza dipende dalle caratteristiche sedimentologiche degli n centimetri precedenti;

secondo, il codice y(c) dipende ovviamente anche da proprie caratteristiche che non si hanno a disposizione , essendo presente l'interruzione che non permette un'analisi diretta. Quindi e' ovvio rilevare che la f(·) nel punto c della sequenza non puo' essere stimata direttamente ma solo attraverso la conoscenza dei valori immediatamente precedenti della sequenza.

Gli aspetti euristici del problema sono dunque :

- determinazione del numero di centimetri noti che possono influenzare il prossimo dato mancante della sequenza

- la ricostruzione della funzione f(·) nel punto mancante.

3.2 IL MODELLO SPERIMENTALE

Nel caso dell'approccio neurale, la classe di funzioni da utilizzare per lo sviluppo della f(·) e' imposta dal tipo di funzione d'attivazione impiegata per determinare l'uscita del singolo neurone. Inoltre si rende necessaria anche una tecnica euristica per la determinazione dell'esatta topologia della rete.

Allo stato attuale non si conoscono tecniche universalmente adattabili per determinare la topologia della rete in maniera precisa. Pertanto e' necessario procedere sperimentalmente. Il problema e' che un numero troppo grande di neuroni "hidden", (nascosti), puo' avere come conseguenza una superclassificazione da parte della rete, nel senso che essa tendera' a discriminare troppo i vari "patterns", (esempi).

Viceversa se il numero e' troppo piccolo, la rete tendera' a generalizzare troppo , classificando come simili patterns fra loro abbastanza differenti.

Il problema qui esaminato si puo' ricondurre parzialmente al ben noto

" Multivariate time series analysis ", che riguarda la possibilita' di studiare il comportamento di serie di dati correlati fra loro da una dipendenza temporale e la predizione dei valori futuri di tali serie basandosi sulla storia passata delle serie stesse, [31].

Consapevoli del fatto che i dati delle serie da noi trattate non dipendono in maniera totale dalla storia passata, abbiamo comunque scelto di sperimentare la suddetta metodologia onde evidenziarne eventuali limitazioni o nuovi sviluppi nel caso dell'approccio con le reti neurali.

Nel nostro caso, le serie sono univariate, poiche' i valori dipendono da una sola variabile spaziale, (la metodologia e' applicabile anche nel dominio dello spazio). Dunque sperimenteremo l'uso di reti neurali per predire i valori futuri di serie spaziali univariate basandoci unicamente sulla storia passata di esse.

Le reti neurali sono sistemi di computazione contenenti molte semplici unita' di calcolo non lineari, interconnesse tra loro. In una rete "feedforward" le unita' o neuroni possono essere partizionate in strati interconnessi mediante dei "links" o pesi.

Una rete i - h - o, possiede i neuroni input, h hidden e o output. Un peso e' associato ad ogni "link", e la rete apprende modificando tali pesi, [31].

Noi utilizziamo tale tipo di rete, in cui i neuroni "hidden" e output realizzano funzioni non lineari della forma $\left(1+\exp\left(-\sum_{i=1}^{m} w_i\, x_i+\theta\right)\right)^{-1}$, dove w_i denota l'i-esimo peso e θ la soglia del neurone ed infine m rappresenta il numero di inputs per il neurone i dallo strato precedente.

Noi usiamo l'algoritmo di Error- Back - Propagation di Rumelhart, [17], per l'apprendimento, usando l'MSE , errore quadratico medio, per giudicare la performance della rete ad ogni iterazione. Dalla serie spaziale esaminata

{ y(c) : 1 ≤ c ≤ N }, otteniamo due insiemi:

un training set Strain = { y(c) : 1 ≤ c ≤ T } , insieme di esempi per addestramento, ed un test set Stest = { y(c) : T ≤ c ≤ N }, insieme di esempi per test.

Un insieme Ptrain di (i+1)-tuple (i primi i elementi input piu' il valore target corretto) ed un insieme Ptest di (i)-tuple sono poi creati da Strain e Stest rispettivamente. L'MSE e' calcolato come $\frac{\sum_{k=1}^{m}\left(u(k)-t(k)\right)^2}{M}$,ove u(k) e t(k) sono output corrente e relativo target desiderato per il k-esimo degli M patterns di training. L'MSE per il test e' analogo.

Ad ogni passo nella fase di training una i-tupla di dati e' presentata alla rete.

Quest'ultima cerca allora di predire il prossimo valore nella sequenza spaziale. L'errore misurato tra i valori calcolati e quelli corretti viene poi retro-

propagato attraverso la rete , modificando i pesi. Se l'MSE eccede un valore di tolleranza, fissato a priori, si procede con una nuova iterazione (ciclo di presentazione di tutti i patterns), alla fine di quella corrente, [18].

I parametri dell'algoritmo sono il "learning rate" o tasso d'apprendimento e il "momento", che descrivono la relativa importanza data al corrente e ai passati valori dell'errore, regolando quindi il grado di modifica da apportare ai vari pesi durante l'apprendimento. Per una performance migliore della rete abbiamo ritenuto di considerare un valore molto piccolo per il "learning rate", (η = 0.01), ed un valore circa medio per il momento (α = 0.6). Il numero di epoche variava tra 10.000 e 20.000 in tutti i casi. Circa il numero di neuroni "hidden", siamo partiti con un valore piccolo, incrementandolo di un'unita' fino al raggiungimento del valore ottimale, che e' risultato essere pari al numero di neuroni input, [24].

Nell'appendice A riportiamo i risultati relativi a due esperimenti particolarmente significativi. Per ulteriori informazioni sul modello di rete neurale, vedere il paragrafo 4.2.

3.3 ANALISI DEI RISULTATI SPERIMENTALI

I due esperimenti riportati in appendice A sono stati da noi scelti, tra tutti quelli effettuati, poiche' contengono da soli tutte le caratteristiche positive e negative relative al comportamento del modello teorico e della rete utilizzati.

Premettiamo la considerazione che la durata delle interruzioni nei segnali analizzati non e' mai nota, ne' tantomeno si puo' fare alcuna ipotesi a riguardo per motivi puramente fenomenologici, [22], [23], [25]. Inoltre le serie analizzate contengono componenti di rumore additivo che possono influire sul comportamento della serie stessa in maniera piu' o meno casuale.

Le caratteristiche rilevate si possono cosi' riassumere:

primo, la rete ha un'ottima performance in fase di training;

secondo, essa si comporta in maniera soddisfacente in fase di testing allorquando la serie di centimetri da ricostruire e' omogenea e corta: omogenea, nel senso che i valori si mantengono analoghi a quelli immediatamente precedenti l'interruzione, senza subire repentini cambiamenti; corta, nel senso che il grado d'affidabilita' della rete tende a diminuire all'aumentare della sequenza da riconoscere. Quest'ultima grave limitazione e' dovuta, secondo il nostro parere, a due eventi: innanzitutto, la rete non riesce in fase di apprendimento ad individuare le caratteristiche omogenee del segnale, rimanendo un sistema "context-sensitive", cioe' dipendente dalle caratteristiche locali del segnale, dando prova in questo caso di una scarsa capacita' di generalizzazione. Inoltre, poiche' la predizione di un valore nuovo si basa su quelli ricavati in precedenza, e' naturale che l'errore si accumuli al crescere dei valori ricostruiti, fino a far degradare eccessivamente la performance totale.

Dunque, questi esperimenti rivelano la scarsa applicabilita' di tale metodologia nel caso in cui i valori della serie in esame non dipendano in maniera completa dalla storia passata di essa. Inoltre, per la ricostruzione di sequenze

complesse, si rende necessaria una metodologia che riesca ad apprendere e a conservare le caratteristiche principali di esse, come periodicita', ricorrenze etc.. Rimandiamo pertanto la risoluzione del problema a successivi esperimenti.

I codici delle reti utilizzate in fase di training e test sono riportati rispettivamente nelle appendici B e C.

4. INDIVIDUAZIONE DEI CICLI DI MILANKOVITCH

4.1 IL METODO CLASSICO

Come abbiamo visto, i metodi di analisi spettrale applicati al segnale stratigrafico consentono di rilevare le frequenze e quindi le periodicita' in esso contenute. Esse, dal punto di vista geologico, rappresentano gli spessori dei cicli principali che formano la successione degli strati. Una fase successiva all'estrazione di tali periodicita' riguarda l'analisi di esse onde trovare eventuali correlazioni con i cicli fondamentali di Milankovitch, [5]. Questi ultimi sono responsabili di un certo grado d'influenza nei cicli deposizionali e sono causati da determinati fenomeni naturali. La seguente tabella evidenzia i fenomeni (colonna 1) e i relativi tempi di durata in migliaia di anni, [4], (colonna 2) :

- precessione breve 18346

- precessione lunga 22056

- obliquita' breve 38202

- obliquita' lunga 49367

- eccentricita' breve 100000

- eccentricita' lunga 410000

Purtroppo non e' possibile correlare in maniera immediata le frequenze estratte dal segnale stratigrafico con le periodicita' di cui sopra, perche' le prime sono quantita' spaziali mentre le seconde sono temporali. Occorre quindi operare delle trasformazioni spazio/tempo per correlare i due gruppi di valori.

Il procedimento che di seguito descriveremo, e' stato recentemente sviluppato da G. Longo, ed e' attualmente utilizzato da alcuni gruppi di geologi, [25].

Per l'applicazione di questo metodo sono necessari i seguenti dati :

1) le periodicita' relative ai cicli di Milankovitch, calcolate numericamente da Berger, [4].

2) le frequenze estratte dal segnale stratigrafico correntemente esaminato

Per rendere possibile il confronto fra 1) e 2) si procede ad elaborare due tabelle, la prima contenente i rapporti fra i valori di 1) tra loro e la seconda contenente i rapporti fra i valori di 2) tra loro. In tal modo si ottengono dei numeri adimensionati che si possono confrontare al fine di trovare eventuali correlazioni, [25].

Nel momento in cui si rileva una corrispondenza, a meno di una certa approssimazione, tra un rapporto xi/xj relativo al gruppo 1) ed uno ti/tj relativo al gruppo 2), la quantita' xi/ti e' il tasso di sedimentazione di un certo spessore xi ogni ti migliaia di anni e quindi rivelera' la presenza dell'i-esimo ciclo , (o frequenza), di Milankovitch nel segnale stratigrafico in esame.

Naturalmente, oltre all'onerosita' del procedimento, il metodo presenta un certo tasso di errore, proporzionale al metodo di analisi spettrale utilizzato e al grado di approssimazione scelto per il confronto delle tabelle, [22].

4.2 L'APPROCCIO CON LE RETI NEURALI

Le note capacita' di classificazione e riconoscimento di caratteristiche da parte delle reti neurali ci hanno spinto a ricercare dei modelli che potessero rendersi alternativi agli approcci classici per l'individuazione dei cicli di Milankovitch all'interno dei segnali spaziali.

La gamma di oggetti di interesse delle reti neurali e' vasta e profonda, coprendo discipline che vanno dalla medicina alla robotica, dall'analisi del parlato al processamento d immagini. Non e' scopo di questa tesi effettuare un'analisi generale delle reti neurali, ne' esplorare oggetti correlati agli aspetti biologici o neurologici di esse. Piuttosto e' nostra intenzione effettuare una breve digressione sulle caratteristiche basilari dei modelli da noi utilizzati in modo da rendere piu' comprensibile l'analisi degli esperimenti effettuati.

Una rete neurale e' uno strumento di analisi modellato sulla struttura ad alto parallelismo del cervello umano. Essa simula una struttura computazionale altamente interconnessa, formata da molti elementi di processo individuali, relativamente semplici, i neuroni, che effettuano calcoli in parallelo. Questi oggetti elementari sono in genere organizzati in gruppi o strati. Questi ultimi possono ricevere inputs (strati input), emettere degli outputs (strati output), o essere inaccessibili ad entrambi gli inputs e gli outputs, avendo solo delle connessioni con altri strati, (strati hidden o interni).

Le reti neurali si caratterizzano attraverso tre elementi fondamentali, [19] :

primo, l'architettura o topologia della rete, che rappresenta il modo particolare in cui gli strati sono interconnessi e ricevono gli inputs e outputs; la connessione fra due neuroni generici avviene mediante un link detto peso.

Secondo, la funzione d'attivazione o di trasferimento scelta per i neuroni, che in analogia con il neurone biologico rappresenta la modalita' di "sparo" del

neurone cioe' di risposta a stimoli esterni. In genere viene scelta la stessa funzione per tutti i neuroni degli strati componenti la rete, ma cio' non e' un vincolo stretto, bensi' una strategia architetturale.

Terzo, l'algoritmo usato per la fase d'apprendimento.

Queste tre caratteristiche possono essere pensate come il livello piu' alto di visione di un modello di rete neurale. E' bene sottolineare che tali aspetti non possono essere sempre considerati indipendenti fra loro, poiche' per esempio, determinate architetture precludono l'utilizzo di determinati algoritmi d'apprendimento o viceversa.

Ultima nota, le connessioni tra i neuroni possono essere rappresentate da valori reali positivi o negativi, in tal caso si parlera' di connessioni rispettivamente eccitatorie ed inibitorie; entrambe influenzano il grado di apprendimento di una rete, [17].

4.2.1 IL MODELLO PERCETTRONE MULTISTRATO CON ALGORITMO DI BACK PROPAGATION.

L'architettura della rete da noi utilizzata e' illustrata in figura 4.1.

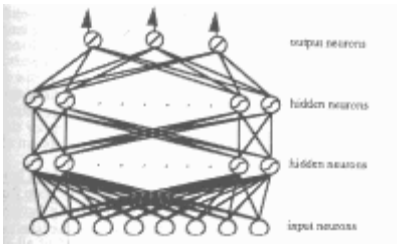


fig. 4.1

I dati fluiscono dallo strato input fino a quello output attraverso successivi spari dei neuroni degli strati interni, [17], [19].

4.2.2 L'ALGORITMO DI APPRENDIMENTO

L'apprendimento back-propagation e' di tipo "supervised", cioe' , ogni output della rete e' confrontato con il valore desiderato corrispondente, in modo da avere un parametro di valutazione sul grado di correttezza della rete ad ogni passo. Esso si basa su due fasi, una denominata "feedforward", cioe' "in avanti", ed una "backward", cioe' "all'indietro", [17].

FASE FEEDFORWARD

Ogni esperimento compiuto mediante una rete neurale richiede una fase preliminare, in cui si effettua una codifica dei parametri input ed output della rete.

Questa codifica ha come fine ultimo la creazione di due pacchetti di esempio patterns, uno per la fase di training, addestramento, ed uno per quella di test.

La scelta di questi due insiemi, nonche' il tipo di codifica costituiscono due degli aspetti piu' delicati di qualsiasi esperimento, pertanto li affronteremo piu' dettagliatamente in seguito.

Supponiamo quindi di avere gia' a disposizione questi due insiemi per un generico esperimento.

All'inizio della fase corrente i patterns sono memorizzati nello strato input. I neuroni di esso semplicemente distribuiscono il segnale ai neuroni del prossimo strato. Osserviamo dalla figura 4.1 che ogni strato e' totalmente connesso al successivo, mediante i link o pesi. Ogni neurone di ciascuno strato effettua due operazioni : inizialmente calcola una somma pesata degli output dei neuroni dello strato immediatamente precedente; cio' costituisce l'input a quel neurone. Per un generico neurone di ciascun strato l'input sara' allora il seguente: Se indichiamo con i , h1, h2 , j , rispettivamente l'indice dei neuroni degli strati input, del primo strato interno, del secondo strato interno e di quello output,

$$net\_input_{h_1} = \sum_i w_{h_1,i}\, o_i = i_{h_1}$$
$$net\_input_{h_2} = \sum_{h_1} w_{h_2 h_1}\, o_{h_1} = i_{h_2} \qquad (5.1)$$
$$net\_input_j = \sum_{h_2} w_{j,h_2}\, o_{h_2} = i_j$$

Con o_x abbiamo indicato l'output del generico neurone dello strato x. Questo output realizza lo sparo del neurone , la seconda operazione, che dipende dalla funzione d'attivazione utilizzata. Quella usata da noi e' la sigmoide, la cui espressione e' la seguente:

$$f(A) = \frac{1}{1+\exp(-A)} \qquad (5.2),$$ ove A e' l'input al neurone.

L'andamento della funzione e' evidenziato in figura 4.2.

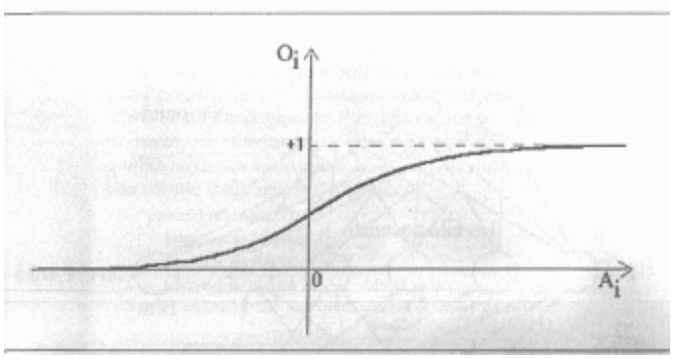


fig. 4.2

Questa funzione in genere e' centrata sul valore 1/2 e cio' da un punto di vista neurale rappresenta una sorta di valore di default per il neurone, il quale spara solo se l'input e' maggiore di zero, mentre resta in quiete, (output 1/2), se

l'input e' nullo, [18]. Per tenere conto di questo fatto, la generica formula delle (5.1) sara' :

$$i_k = \sum_n w_{kn}\, o_n - B_k \qquad (5.3).$$

Il termine Bk rappresenta la soglia della k-esima unita', cioe' il punto dove centrare la sigmoide; chiameremo "bias", ossia "tendenza", tale soglia, per esprimere il fatto che in assenza di stimoli l'unita' tende ad assumere quel valore.

I bias dei diversi neuroni possono essere diversi fra loro e dipendono dall'applicazione, cioe' dalla funzione che la rete deve saper svolgere. In altri termini, il valore Bk dovra' essere appreso alla stessa stregua dei pesi w; questo suggerisce di considerare il bias come il peso di una connessione fittizia verso un neurone fantasma che vale sempre 1. Quindi, supponendo di avere N neuroni nel livello x della rete, il bias sara' realizzato attraverso un N+1-esimo neurone, il cui valore e' fisso ad 1 e le cui connessioni verso gli altri neuroni rappresentano l'opposto del loro bias, [18].

Pertanto la formula generica delle (5.1) diviene:

$$i_k = \sum_{n=1}^{N+1} w_{kn}\, o_n \qquad (5.4),$$ con l'unica differenza che la sommatoria ha un termine in piu'.

FASE BACKWARD

Cio' che si e' potuto notare in precedenza, e' che la rete e' priva di connessioni cicliche, vale a dire che le connessioni fra i neuroni sono unidirezionali. Cio' che e' ciclico e' il meccanismo di apprendimento : infatti, per ottenere una giusta combinazione dei pesi, e' necessaria una formula che operi per epoche, vale a dire che aggiusti i pesi gradualmente, per piccoli passi,

ciclando piu' volte sui patterns da apprendere. Il meccanismo e' il seguente : dato un insieme di patterns nella forma {X,D}, ove X e' l'input alla rete e D l'output desiderato, la rete inizialmente prova ad eseguire l'input X e vede quale output Y salta fuori con gli attuali pesi; questo viene confrontato con D e si misura cosi' l'errore commesso; quest'ultimo viene "retropropagato" nella rete, aggiustando i pesi. Il tutto viene ripetuto fino ad esaurimento dei patterns e cio' costituisce un'epoca di apprendimento. A questo punto si ripetono le due fasi dell'algoritmo etc....

Le epoche vengono ripetute fino a che l'errore commesso non risulta inferiore ad un valore scelto a priori, [17], [18].

L'errore per un neurone generico dello strato output e' definito dunque come la differenza tra il valore corrente e il valore desiderato : $E_j = D_j - Y_j$ (5.5), mentre l'errore globale alla fine di un ciclo feedforward e' $E = \frac{1}{2}\sum_j \left(D_j - Y_j\right)^2$ (5.6).

Se consideriamo l'andamento di un singolo peso della rete durante diverse epoche d'apprendimento, avremo un grafico piu' o meno simile al seguente :

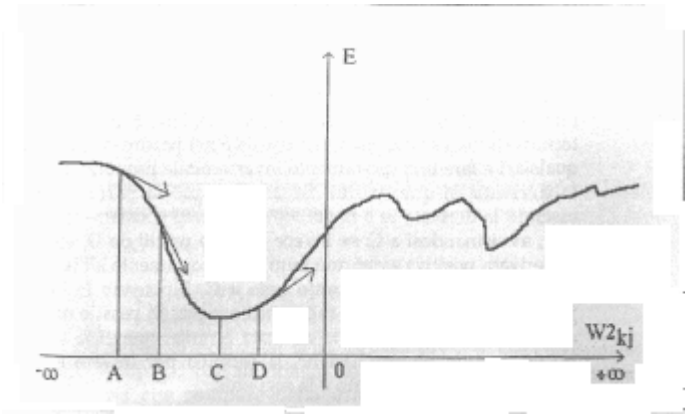


fig. 4.3

Il valore di w corrispondente al minimo di quella curva (punto C in figura 4.3), e' proprio quello che cerchiamo : cioe' se tutti i pesi della rete fossero corrispondenti al rispettivo minimo, la rete avrebbe appreso correttamente. Il problema e' dunque quello di trovare il valore minimo di ogni peso, cioe' trovare il minimo di una funzione in uno spazio N-dimensionale. La tecnica usata per questo scopo e' quella del "gradiente discendente". L'idea e' quella di partire da un certo punto, ad esempio il punto A della figura 4.3 e calcolare qui la derivata della funzione errore. A seconda del valore da essa assunto si proseguira' in avanti o a ritroso attraverso i punti B e D fino a raggiungere il minimo desiderato, in cui la derivata e' nulla o quasi, [32].

Dunque le quantita' utilizzate per questo procedimento sono :

$$\begin{array}{l} \delta = f'(x)(D-Y) \quad \text{per lo strato output} \\ \delta' = f'(x)\sum w\delta \quad \text{per gli altri strati} \end{array} \quad , \qquad (5.7)$$

ove x e'l'input al generico neurone.

Tali quantita' costituiscono la base per l'aggiornamento dei pesi della rete insieme ad altri due coefficienti variabili, rispettivamente il tasso d'apprendimento eta ed il termine momento alpha, entrambi assumenti valori tra 0 ed 1. La formula per l'aggiornamento del peso generico sara' allora :

$$W_{ji}(new) = W_{ji}(old) + eta * \delta_{\,i} o_j + alpha * \Delta w_{ji}(old) \qquad (5.8),$$

ove $\Delta w_{ji}(old)$ rappresenta la variazione precedente del peso, [17].

Osserviamo che all'inizio dell'algoritmo i pesi sono scelti "random" in un intervallo fissato, generalmente dell'ordine di 10^-1.

Inoltre, la scelta dei parametri eta ed alpha e' uno dei punti critici di un esperimento e di solito e' effettuata in maniera sperimentale. Il ruolo di essi nell'algoritmo sara' descritto di seguito.

4.2.3 CONSIDERAZIONI GENERALI SUL MODELLO.

La struttura della rete multistrato e' capace di generare le rappresentazioni interne che permettono di classificare le regioni input che o si intersecano fra loro o sono disgiunte.

Il minimo numero di strati richiesti per realizzare un'arbitraria operazione di "mapping" e' tre, ma nel nostro caso abbiamo utilizzato uno strato interno in piu'.

Considerando gli strati superiori a quello input individualmente, possiamo

evidenziare piu' dettagliatamente il ruolo da essi esercitato nell'esperimento: il primo strato forma gli iperpiani che partizionano effettivamente lo spazio delle caratteristiche in varie sottoregioni, il secondo realizza l'AND di questi piani insieme per formare regioni chiuse e l'ultimo strato realizza l'OR di queste per creare un insieme di regioni disgiunte, permettendone la separazione e dunque la classificazione, [32], [33].

Il ruolo finale della rete addestrata e' ovviamente fornire la risposta corretta a dati nuovi, che cioe' non fanno parte del training set, bensi' del test set; in pratica la struttura interna della rete dovrebbe essere in grado di **generalizzare** la classificazione in base alle informazioni acquisite durante l'apprendimento.

Il grado di generalizzazione e' in pratica la differenza tra la capacita' di classificare i patterns del training set e quella di classificare i patterns del test set.

Essa dipende dai seguenti fattori , [33] :

1) la taglia della rete : il numero di parametri liberi nella rete e' un punto di congiunzione tra la desiderata accuratezza, l'efficienza computazionale ed il problema potenziale della "superclassificazione" dei dati. Se tale numero e' troppo piccolo, la rete sara' incapace di apprendere l'informazione rilevante del training set e quindi la rete avra' una scarsa capacita' di generalizzazione. Viceversa se il numero e' troppo grande, la rete inizia a memorizzare i dettagli del training set e

fallira' nell'apprendere le regole fondamentali, ottenendo comunque una scarsa generalizzazione. In pratica, il numero di neuroni degli strati interni puo' essere visto come il numero di iperpiani in cui viene diviso lo spazio multidimensionale. Se un tale numero e' troppo piccolo, altrettanto ridotto sara' il numero di iperpiani con cui separare lo spazio della caratteristiche, ottenendo una scarsa classificazione; se il numero e' troppo grande, altrettanto sara' quello degli iperpiani, causando un'eccessiva separazione dello spazio.

2) il range dei valori dei pesi : durante il processo d'apprendimento, i valori dei pesi crescono con il numero di presentazioni dei patterns. E' stato dimostrato [32], che porre dei limiti ai valori iniziali dei pesi, puo' migliorare la capacita' di generalizzazione delle reti MLP, (Multi Layer Perceptrons).

3) il numero di cicli di training : il problema della generalizzazione e' solitamente affrontato inizializzando una rete usando piccoli pesi "random" e bloccando il processo di apprendimento dopo un certo numero di cicli. Cio' affinche' il training set non sia appreso in eccessivo dettaglio. Un apprendimento troppo prolungato portera' a grossi valori dei pesi che potrebbero provocare una scarsa capacita' di generalizzazione.

4) la presentazione dei patterns : e' importante seguire una via di mezzo tra il numero di patterns con un valore output elevato e quelli con un output piu' basso, altrimenti la rete cerchera' di apprendere i patterns che producono solo un particolare output. In genere una soluzione e' ripetere piu' volte nello stesso ciclo la presentazione dei patterns con valore piu' basso.

5) la codifica input/output : il metodo di codifica input/output e' molto importante. Un'appropriata strategia di codifica puo' aumentare notevolmente la capacita' di apprendimento, nonche' di generalizzazione. In particolare, si e' visto che le reti MLP hanno grande difficolta' se due elementi sono codificati in modo da rientrare in due classi differenti nello spazio input, ma sono molto vicini, in termini di distanza metrica, tra loro nello spazio output.

6) il tipo di apprendimento : in un algoritmo di back-propagation, le derivate della funzione errore sono sommate su tutti i patterns. Nell'apprendimento di tipo "batch", i pesi sono aggiornati dopo la presentazione di tutti i patterns, mentre in quello di tipo "on-line", dopo ogni pattern. L'esperienza dimostra che se il tasso d'apprendimento eta e' sufficientemente piccolo, non vi e' grossa differenza tra i due metodi in termini di rendimento.

7) scelta del tasso d'apprendimento e del fattore momento : in genere, piu' alto e' il tasso d'apprendimento eta, maggiori sono i cambiamenti dei valori dei pesi. Ovviamente, in teoria sarebbe preferibile avere un valore di eta piu' grande possibile in modo da ottenere un apprendimento piu' rapido possibile, ma cio' spesso provoca il blocco della rete in un minimo locale della funzione errore causando uno scarso rendimento in termini di generalizzazione. Un modo per aumentare la percentuale d'apprendimento ad ogni passo evitando simili rischi, e' introdurre il termine momento alpha nella formula di aggiornamento dei pesi. Esso effettivamente ha il ruolo di un filtro, che attenua il passaggio delle variazioni ad alta frequenza della superficie errore nello spazio dei pesi. Cio' e' particolarmente utile quando tale superficie e' formata da grossi dislivelli. La scelta dei valori numerici per questi due parametri e' frutto di esperienza personale e di buon senso, non essendo disponibile alcun criterio oggettivo per tale decisione.

5. DESCRIZIONE DELL'ESPERIMENTO

In questo capitolo descriveremo l'esperimento condotto su un segnale reale, procedendo per gradi: inizieremo con la descrizione del segnale di test, riportando il grafico della sequenza rappresentante la successione montuosa; procederemo poi, descrivendo la procedura di costruzione dei segnali sintetici, costituenti l'insieme di esempi con cui addestrare la rete; riporteremo successivamente i risultati sia dell'addestramento che della fase di test al variare dei parametri della rete. Infine effettueremo un confronto con il metodo dei rapporti, descritto in precedenza, dimostrando la buona riuscita dell'esperimento.

5.1 IL SEGNALE REALE

La sequenza riportata in figura 5.1, si riferisce alla rappresentazione codificata di una successione estratta dal Monte Tobenna (SA), di durata pari a 55.39 metri di parete, campionata a intervalli di 1 cm., per un totale di 5539 dati numerici.

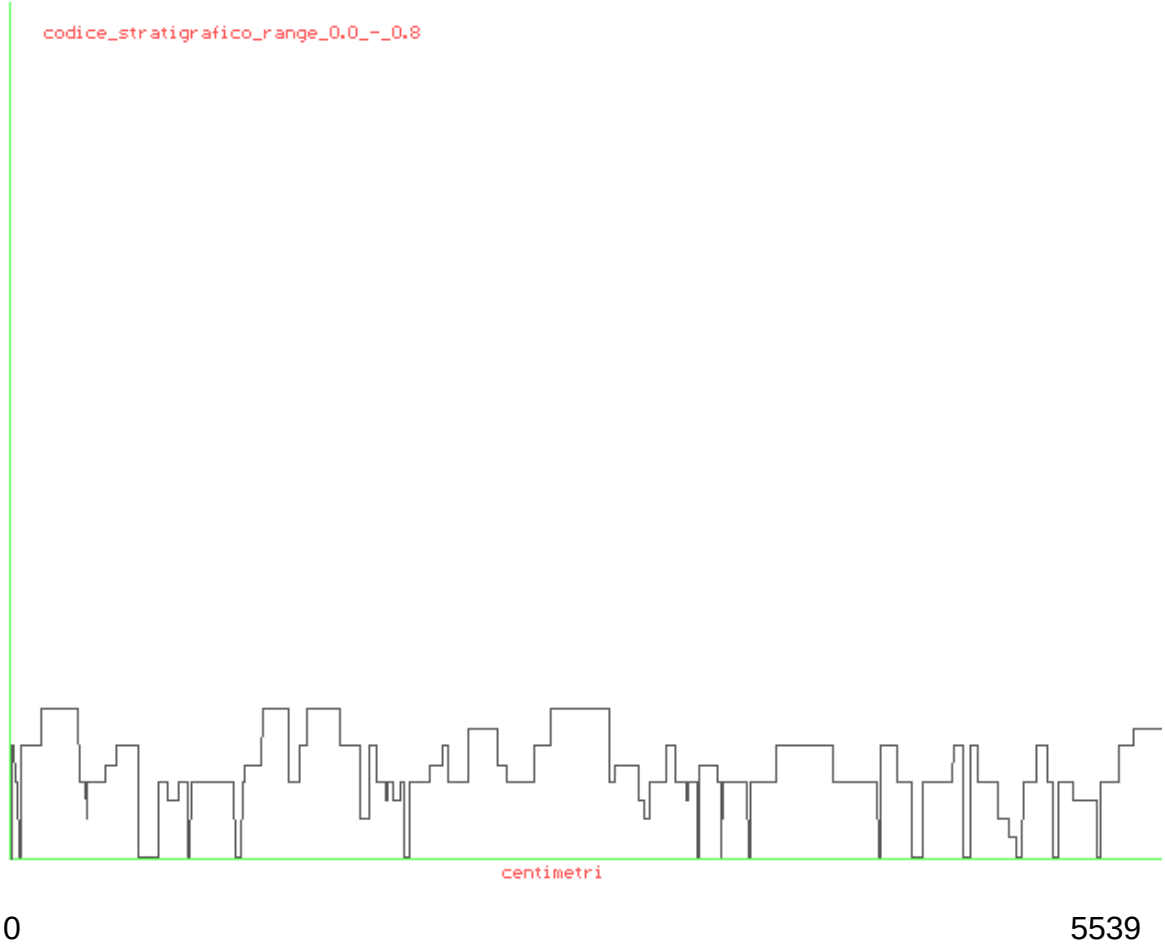


fig. 5.1 - grafico della sequenza del monte Tobenna

Il "range" delle ampiezze dell'onda quadra e' [0.0 , 0.8], [5].

5.2 LA FASE DI PREPROCESSAMENTO DEI DATI

Una fase molto delicata di un qualsiasi esperimento realizzato con una rete neurale, e' la preparazione dei dati di addestramento, o costruzione del training set, per la rete. Gran parte del merito per la riuscita dell'esperimento si deve in genere alla scelte ad hoc effettuate in questa fase preliminare, [33].

Nel nostro caso, dovevamo insegnare alla rete a riconoscere la presenza di periodicita', nascoste da rumore, all'interno di un segnale ed a classificarle correttamente, scegliendo tra sei possibili classi di appartenenza : le sei frequenze di Milankovitch, ottenute dalle sei periodicita' enumerate in precedenti capitoli.

Denoteremo tali sei frequenze nel seguente modo :

CARATTERISTICA	**PERIODICITA' Pi**	**FREQUENZA: Fi = 1/Pi**
eccentricita' lunga	410.000	2.439E-06
eccentricita' breve	100.000	1E-05
obliquita' lunga	49367	2.025E-05
obliquita' breve	38202	2.617E-05
precessione lunga	22056	4.534E-05
precessione breve	18346	5.451E-05

Ricordiamo che i periodi Pi sono espressi in KA, cioe' migliaia di anni.

A questo punto la fase di costruzione dei patterns di training e' stata la seguente :

Notazioni :

S(c) : segnale reale da classificare, c = 1,...,5539.

S(t) : segnale temporale sintetico.

Δc : intervallo di campionamento di S(c), Δc = 1 cm.

N0 : numero di campioni del segnale reale, N0 = 5539.

Fi : frequenza temporale di Milankovitch, i = 1,...,6.

Fc : frequenze spaziali significative del segnale reale, c = 1,...7, (cap. 2).

Pi : periodicita' temporali di Milankovitch, Pi = 1/Fi, i = 1,...,6.

Pc : periodicita' spaziali del segnale reale, Pc = 1/Fc, c = 1,...,7, (cap. 2).

α : coefficiente di correlazione tempo/spazio, α = Pi/Pc.

Δt : intervallo di campionamento di S(t), $\Delta t = \alpha$.

5.2.1 GENERAZIONE DI SEGNALI SINTETICI

L'idea di base dell'addestramento e' far apprendere la rete su segnali discreti sintetici, cioe' generati artificialmente, che abbiano le seguenti caratteristiche:

- devono contenere le diverse combinazioni delle sei frequenze di Milankovitch, in modo tale che la rete possa riconoscere la presenza di una di queste combinazioni nel segnale reale.

- devono possedere rumore, "noise", additivo, per mascherare le sinusoidi create sulle diverse frequenze Fi.

- devono avere lo stesso numero di punti del segnale reale in questione.

- devono avere una forma simile al segnale reale, cioe' onde quadre, con ampiezze analoghe a quelle del segnale originale.

- devono essere campionati con lo stesso tasso del segnale reale; in tal caso, poiche' tali segnali sintetici sono creati su frequenze temporali, mentre quello reale contiene frequenze spaziali, e' necessario trovare un coefficiente di correlazione spazio/tempo α, che permetta l'associazione di uno stesso tasso di campionamento, [22], [23], [25].

Per soddisfare tali esigenze, abbiamo provveduto alla seguente procedura: abbiamo effettuato un'analisi spettrale, con il metodo di Scargle descritto nel cap. 2, [6], del segnale reale, da cui abbiamo ottenuto un set di frequenze caratteristiche del segnale.

Abbiamo preso in considerazione la frequenza Fc in corrispondenza del picco massimo dello spettro del periodogramma ed abbiamo ipotizzato che la corrispondente periodicita' Pc = 1/Fc, fosse associata alla periodicita' temporale Pi corrispondente alla frequenza piu' bassa Fi di Milankovitch. Quest'ipotesi e' la chiave dell'esperimento, poiche', nel caso sia corretta, tale risultera' anche la classificazione della rete; infatti da quest'associazione si puo' risalire al tasso con cui campionare i segnali sintetici, poiche' in generale $\alpha = \Delta t = Pi/Pc$. Ovviamente solo nel caso in cui i due intervalli di campionamento sono identici, sara' possibile da parte della rete confrontare correttamente i due tipi di segnali. Nel caso in cui tale ipotesi fallisca, si reitera tutto il procedimento associando la stessa frequenza di Milankovitch al prossimo picco immediatamente piu' basso del precedente dello spettro del periodogramma. A questo punto osserviamo che, in primo luogo, fissato α, esso sara' uguale, a meno di piccole oscillazioni trascurabili, per tutte le altre frequenze; inoltre non e' necessario continuare il procedimento per tutte le frequenze Fc del segnale reale, ma e'sufficiente fermarsi a quelle relative ai primi due picchi dello spettro, poiche' e' scarsamente probabile che frequenze spaziali piu' alte possano associarsi alle frequenze temporali piu' basse. Inoltre il motivo della scelta della frequenza Fi piu' bassa come riferimento e' motivata dalla sua maggiore ricorrenza all'interno delle successioni esaminate.

Nel nostro caso dunque, la corretta classificazione si e' avuta associando la frequenza temporale F1 a quella spaziale Fc = 9.04E-04 corrispondente al secondo picco piu' alto nello spettro del periodogramma, avente periodicita' Pc = 1106 cm. Il tasso di campionamento risultante e' stato dunque:

$\Delta t = \alpha = P1/Pc = 0.000/1106 = 370.036$.

La prossima scelta e' stata il numero di punti da assegnare ai segnali sintetici in modo che fosse uguale a quello del segnale reale, tenendo presente Δt.

Se indichiamo con N il numero di punti dei segnali artificiali, si e' avuto:

N = No * Δt = 2049630.

Per poter inserire del rumore additivo, abbiamo realizzato una procedura che, fissata una costante ε compresa tra 0 e 0.5 ed alcuni valori generati in maniera casuale, aggiungeva alle varie sinusoidi del segnale una serie di valori numerici, ottenuti da una distribuzione normale con media $\mu = 0$ e varianza $\sigma^2 = 1$, se il numero "random" risultava minore di ε. L' algoritmo di generazione era dunque il seguente :

ε = 0.3

N = 2049630

Δt = 370.036

```
for(t = 1; t < N; t = t + Δt)
 {
   S(t) = 1 + sin(2πF1t) + .....+ sin(2πF6t)
   if(random() < ε)
     {
       S(t) = S(t) + gauss(random()) + .....+ gauss(random())
     }
 }
```

La funzione **random()** restituisce un valore casuale diverso, compreso tra 0 ed 1, ogni volta che viene chiamata. La funzione **gauss(x)** calcola il valore della distribuzione normale al variare di x. Da notare la traslazione in ampiezza delle sinusoidi, ottenendo valori tra 0 e 2 anziche' tra -1 ed 1; cio' per evitare valori negativi nel segnale. Inoltre i puntini sospensivi stanno a significare il diverso numero di frequenze di Milankovitch incluse nel segnale, nonche' la diversa

quantita' di rumore aggiunto. Il numero di segnali sintetici generato e' stato pari al numero di possibili combinazioni delle 6 frequenze di Milankovitch, cioe' NS = 2^6 = 64. Infine, per soddisfare l'ultima richiesta, cioe' che il segnale sintetico fosse espresso sottoforma di onda quadra con ampiezze analoghe a quelle del segnale reale, si e' provveduto ad una normalizzazione variabile dei valori di S(t). Il grafico in figura 5.2 si riferisce al segnale sintetico contenente le sei frequenze di Milankovitch con rumore additivo generato da una sola gaussiana.

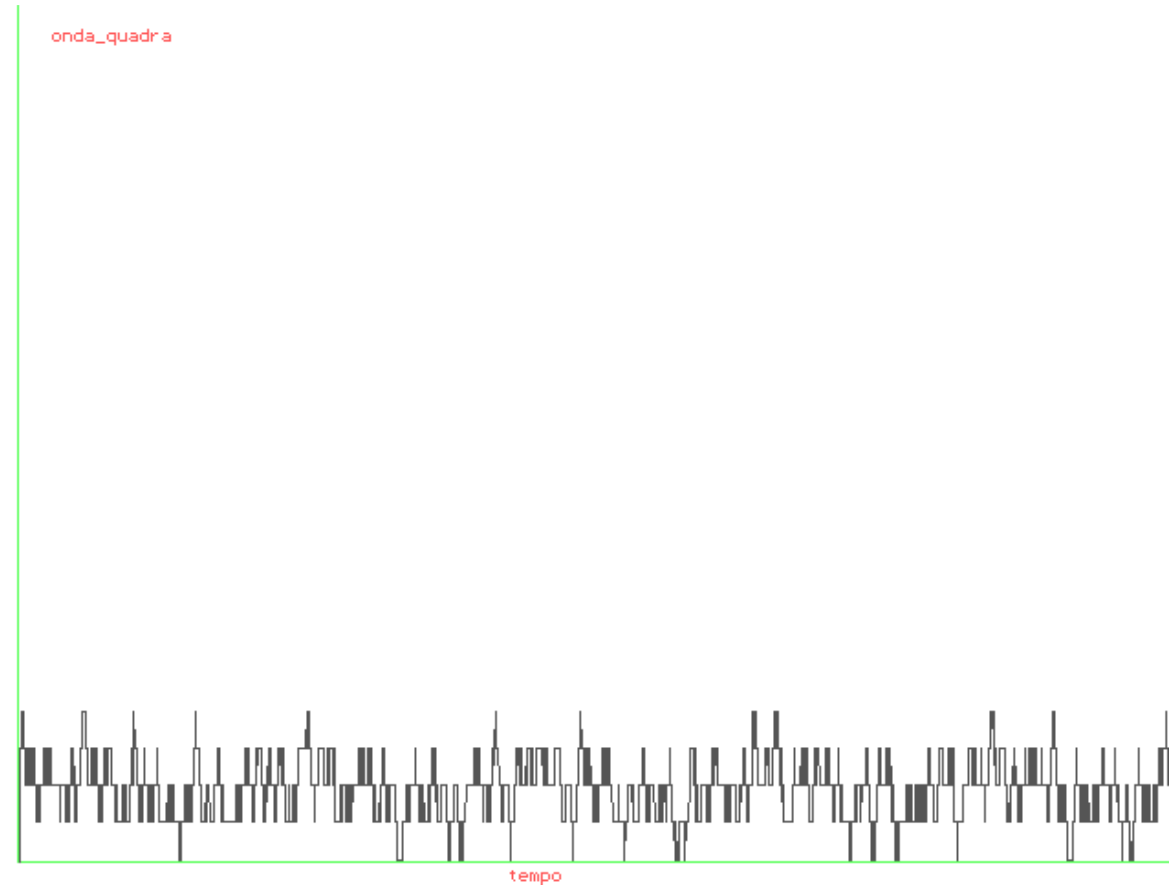


fig. 5.2 - segnale sintetico contenente le 6 frequenze di Milankovitch

5.2.2 COSTRUZIONE DEL TRAINING SET

La fase successiva alla generazione dei segnali sintetici prevede la loro manipolazione in modo da creare il training set, cioe' l'insieme di patterns con cui addestrare la rete.

La difficolta' a questo punto della procedura nasce dalla consapevolezza che il modello di rete utilizzato non e' invariante alla traslazione. Cio' significa che non e' possibile suddividere semplicemente il segnale input in porzioni di una certa lunghezza e farle apprendere presentandole una di seguito all'altra, poiche' la rete non sarebbe in grado di memorizzare le informazioni salienti presenti nel segnale con continuita', [14], [15].

Peraltro una suddivisione del segnale e' necessaria, poiche' e' impensabile utilizzare migliaia di neuroni input per immagazzinare tutta la sequenza in un colpo solo.

L'idea e' dunque di suddividere il segnale in sottosequenze che contengano tutte le informazioni preziose ai fini dell'apprendimento, ma che siano ragionevolmente ridotte come dimensione, facendo attenzione a non renderle troppo corte, altrimenti lo spazio input della rete avrebbe un numero di dimensioni insufficiente per un valido apprendimento.

La procedura risultante e' la seguente :

suddividiamo il segnale campione in sottosequenze di uguale dimensione, tranne al piu' l'ultima porzione, che rispettino le seguenti condizioni:

- il numero di punti di ciascuna sottosequenza deve essere almeno pari alla lunghezza del periodo in centimetri Pc preso come riferimento per il calcolo del coefficiente di correlazione α. Cio' perche' in tal modo ognuna conterra' le informazioni complete relative a tutte le periodicita' ipotizzate nel segnale. Ricordiamo infatti che avendo preso come riferimento, per l'ipotesi di partenza di

tutta la procedura, il periodo Pc piu' grande del segnale, in esso saranno contenuti tutti gli altri eventualmente presenti.

- ogni sottosequenza contiene una percentuale pari al 50% di "overlap", cioe' contiene per meta' punti nuovi del segnale e per meta' la parte finale della sottosequenza precedente. Cio' per fornire maggiore continuita' ai vari frammenti del segnale totale.

- ogni sottosequenza deve essere codificata in modo da :

fornire una rappresentazione in frequenza del segnale;

ridurre il rumore di fondo;

ridurre le dimensioni dello spazio input della rete.

Per poter soddisfare tutte le esigenze analizzate, abbiamo elaborato la seguente procedura:

Sia NS il numero delle sottosequenze costituenti il segnale e NP = Pc la dimensione di ciascuna di esse.

- sottraiamo la varianza del segnale ad ogni sottosequenza Si.

- per ogni sottosequenza Si, i = 1,...,NS, calcoliamo T = DFT_{NP}(Si), (trasformata di Fourier discreta), nella banda [0 , 1/2[.

- normalizziamo i valori di ciascuna trasformata nell'intervallo [-1 , 1].

- da ogni sottosequenza Si ricaviamo un insieme di 6 elementi, che sono i valori normalizzati dello spettro di Fourier corrispondenti alle 6 frequenze di Milankovitch, F1,....,F6, cosituendo una sorta di banco di filtri, tarati sulle 6 frequenze da riconoscere.

- infine componiamo le sottosequenze a gruppi di due, (secondo overlap), ottenendo quelli che denoteremo i "frames" di un segnale: ogni frame conterra' 12 elementi, rappresentanti l'unione di due sottosequenze adiacenti, (la prima con la seconda, la seconda con la terza etc...); il motivo di questo secondo overlap e' dovuto alla necessita' di ampliare lo spazio input della rete per migliorare le capacita' di apprendimento.

Nel nostro esempio abbiamo ottenuto:

numero di sottosequenze NS = (N0 / Pc * 2) - 2 = 9.

numero di punti di ciascuna sottosequenza NP = 1108.

numero di frame NF = 8

numero di componenti di ciascun frame NC = 12.

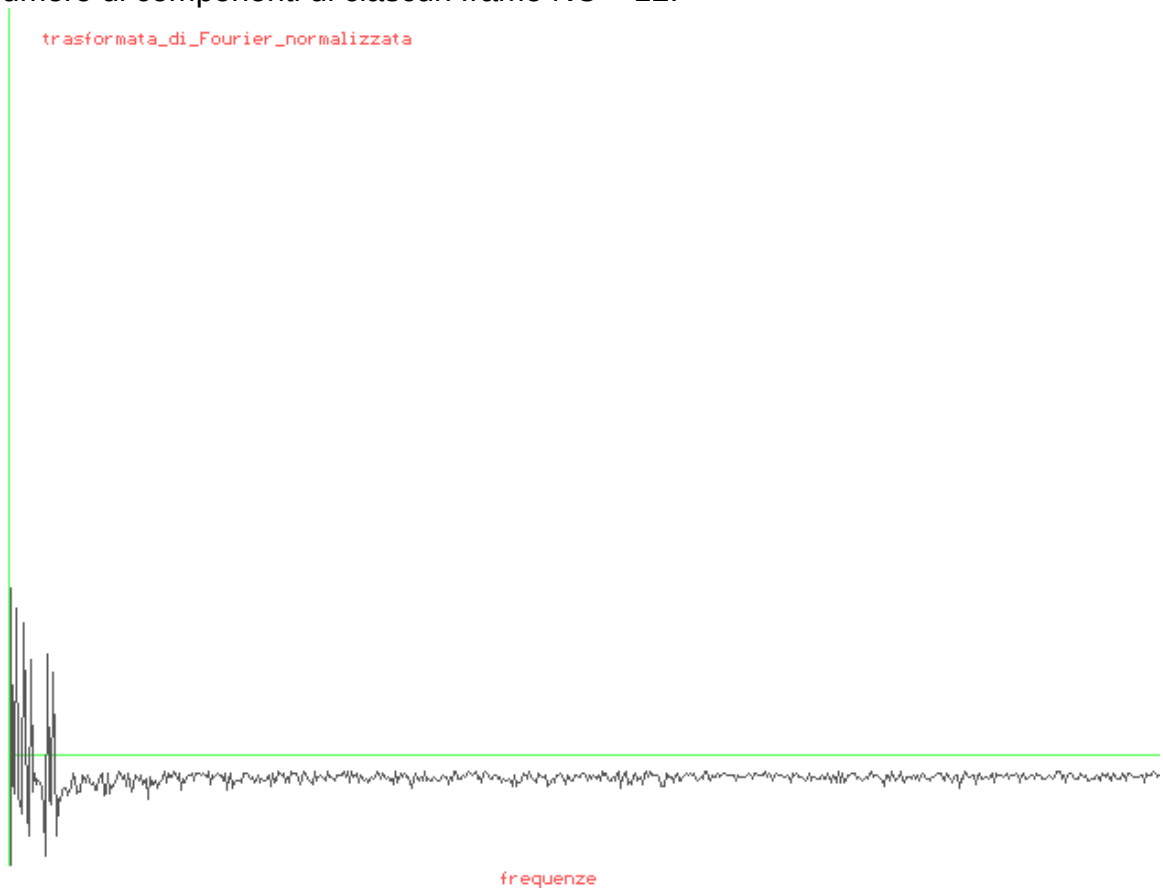


fig. 5.3 - DFT della prima sottosequenza del segnale sintetico di fig. 5.2 nella banda [0,1/2]. I valori di ampiezza sono normalizzati tra -1 e 1.

L'ultima fase riguarda la scelta degli esempi di training per la rete.

Precisiamo che nel seguito il nome "pattern" sara' riferito al singolo segnale, per cui ogni pattern, sia esso di training o di test, sara' composto da 8 frames, ciascuno contenente 12 elementi.

Nel nostro esempio abbiamo utilizzato il training set composto dai seguenti segnali :

SEGNALE	**NUMERO FREQUENZE**	**FREQUENZE (F1,....F6)**
sintetico	4	F1, F2,F3,F4
"	2	F1, F2
"	5	F2,F3,F4,F5,F6
"	2	F1,F6
"	5	F1,F3,F4,F5,F6
"	2	F2,F4
"	4	F1,F2,F5,F6
"	2	F3,F4
"	4	F1,F3,F5,F6
"	2	F3,F5
"	4	F1,F3,F4,F5
"	2	F4,F5
"	2	F5,F6
"	3	F1,F2,F3
"	3	F1,F3,F5
"	3	F1,F4,F6
"	3	F1,F5,F6
"	3	F2,F3,F4
"	5	F1,F2,F4,F5,F6
"	3	F2,F3,F5
"	3	F2,F3,F6
"	3	F2,F4,F6

"	3	F4,F5,F6
"	3	F1,F4,F5
"	4	F1,F2,F4,F6
"	4	F2,F3,F4,F5
"	4	F2,F4,F5,F6
"	2	F2,F3
"	4	F3,F4,F5,F6
"	5	F1,F2,F3,F4,F5

L'ordine di presentazione dei patterns e' stato realizzato in modo da fornire variabilita' nell'apprendimento.

Il test set e' invece formato dai seguenti patterns :

SEGNALE	**NUMERO FREQUENZE**	**FREQUENZE**
Tobenna (reale)	5	F1,F2,F3,F4,F5
sintetico	6	F1,F2,F3,F4,F5,F6
"	1	F1
"	2	F1, F5
"	3	F2,F4,F5
"	4	F1,F2,F3,F5

Riguardo al segnale reale Tobenna, le frequenze indicate sono quelle estrapolate con i metodi classici in dotazione ai geologi, fra cui il metodo dei rapporti, discusso nel cap. 2, [5], [22], [23], [25].

Osserviamo infine che la stessa codifica utilizzata per il training set, e' stata applicata al test set.

5.3 TRAINING E TEST DELLA RETE

Conclusa la fase di preparazione e codifica del training e test set, l'ultimo problema riguarda l'inizializzazione dei parametri della rete; ossia:

1. scelta della topologia ottimale, (numero di strati e di neuroni di ciascun strato).

2. assegnazione dei valori ottimali ai seguenti parametri :

- tasso d'apprendimento eta

- momento alpha

- intervallo entro cui generare i pesi iniziali della rete

- limite di tolleranza dell'errore quadratico medio per ogni iterazione

- numero massimo di iterazioni da eseguire sul training set.

Come abbiamo gia' sottolineato, non vi sono leggi definite per risolvere in maniera oggettiva tali problemi, pertanto abbiamo proceduto secondo le seguenti modalita': Abbiamo realizzato alcune decine di esperimenti, variando di volta in volta alcuni parametri tra quelli enumerati in precedenza. Riportiamo di seguito le tabelle relative solo agli esperimenti piu' significativi.

OSSERVAZIONE

Nelle tabelle di seguito illustrate, abbiamo fissato i valori dei seguenti parametri, la cui ottimalita' e' risultata dagli altri esperimenti non riportati per brevita':

- intervallo valori di inizializzazione pesi : da varie esperienze, abbiamo scelto l'intervallo [- 0.2 , 0.2].

- valore di tolleranza dell'errore output : e' stato fissato a 0.02, onde evitare un numero troppo elevato di iterazioni.

TABELLA 1

Si riferisce ai valori dell'errore quadratico medio ed al numero d'iterazioni durante la fase di apprendimento, al variare di alcune topologie piu' significative della rete e dei valori attribuiti al tasso d'apprendimento ed al fattore momento, rispettivamente indicati con, EMS TRAIN, #_CICLI, TIPO DI RETE, ETA ed ALPHA; per la voce TIPO DI RETE, si e' scelta l'espressione "in-hid1-hid2-out", che indica il numero di neuroni rispettivamente degli strati input, primo strato interno, secondo strato interno ed output.

INDICE ESPER.	TIPO DI RETE	ETA	ALPHA	#_CICLI	EMS TRAIN
1	12 - 6 - 6 - 6	0.001	0.1	77'000	0.013117
2	"	0.04	0.05	5'900	0.009991
3	"	0.005	0.05	30'000	0.010023
4	"	0.002	0.05	34'400	0.016027
5	"	0.001	0.05	60'000	0.016196
6	"	0.002	0.1	32'000	0.016144
7	12 - 8 - 7 - 6	0.002	0.075	60'000	0.015989
8	"	0.005	0.95	1'500	0.009851
9	"	0.001	0.1	30'000	0.016000
10	"	0.002	0.1	27'600	0.011466
11	"	0.001	0.05	37'700	0.015389
12	"	0.002	0.05	32'000	0.010860
13	12 -10 -0 - 6	0.001	0.1	10'300	0.020266
14	"	0.04	0.05	5'800	0.009977
15	"	0.005	0.05	30'000	0.011278

Dalla tabella si puo' notare come al crescere di eta ed alpha, il numero di iterazioni tende a diminuire drasticamente e di conseguenza anche il tempo di

convergenza. Cio' pero' non sempre e' indice di migliore apprendimento, poiche' in questi casi la stabilizzazione della funzione errore in un minimo locale, anziche' assoluto e' piu' probabile. E' buona norma mantenere eta sufficientemente basso, in modo che l'apprendimento sia meno rapido, ma piu' sicuro, [17], [32], [33].

TABELLE 2

Si riferiscono alla fase di test, in cui sono riportati l'errore quadratico medio output (EMS TEST), le frequenze da riconoscere (FREQUENZE TARGET) e le frequenze riconosciute (FREQUENZE OUTPUT) per i sei segnali (patterns) del test set. La voce INDICE ESPERIM. si riferisce al tipo di esperimento descritto nella tabella 1.

TABELLA 2.1

SEGNALE	INDICE ESPERIM.	EMS - TEST	FREQUENZE TARGET	FREQUENZE OUTPUT
Tobenna	1	0.124958	F1,F2,F3,F4, F5	F1,F2,F3,F4, F5
"	2	0.186130	"	F1,F2,F3,F4
"	3	0.266820	"	F1,F2
"	4	0.180629	"	F1,F2,F4,F5
"	5	0.181947	"	F1,F2,F4,F5
"	6	0.180967	"	F1,F2,F4,F5
"	7	0.202568	"	F1,F2,F4,F5
"	8	0.256557	"	F1,F2,F4
"	9	0.204840	"	F1,F2,F4
"	10	0.223489	"	F1,F2,F4,F5
"	11	0.205026	"	F1,F2,F4,F5
"	12	0.234876	"	F1,F2,F4,F5
"	13	0.253668	"	F1,F2,F4
"	14	0.289350	"	F1,F2
"	15	0.293071	"	F1,F2

TABELLA 2.2

Si riferisce al segnale sintetico contenente 6 frequenze.

SEGNALE	INDICE ESPERIM.	EMS - TEST	FREQUENZE TARGET	FREQUENZE OUTPUT
sintetico	1	0.007166	F1,F2,F3,F4, F5,F6	F1,F2,F3,F4, F5,F6
"	2	0.053737	"	F1,F2,F4,F5, F6
"	3	0.066274	"	F1,F2,F4,F5, F6
"	4	0.013099	"	F1,F2,F3,F4, F5,F6
"	5	0.015668	"	F1,F2,F3,F4, F5,F6
"	6	0.013460	"	F1,F2,F3,F4, F5,F6
"	7	0.000092	"	F1,F2,F3,F4, F5,F6
"	8	0.105719	"	F1,F2,F4,F5, F6
"	9	0.000155	"	F1,F2,F3,F4, F5,F6
"	10	0.000053	"	F1,F2,F3,F4, F5,F6
"	11	0.000082	"	F1,F2,F3,F4, F5,F6
"	12	0.000038	"	F1,F2,F3,F4, F5,F6
"	13	0.000764	"	F1,F2,F3,F4, F5,F6
"	14	0.000004	"	F1,F2,F3,F4, F5,F6
"	15	0.000005	"	F1,F2,F3,F4, F5,F6

TABELLA 2.3

Si riferisce al segnale sintetico contenente la prima frequenza.

SEGNALE	INDICE ESPERIM.	EMS - TEST	FREQUENZE TARGET	FREQUENZE OUTPUT
sintetico	1	0.028888	F1	F1
"	2	0.219442	"	F1,F2,F3
"	3	0.272774	"	F1,F2,F3,F4
"	4	0.083259	"	F1,F3
"	5	0.087828	"	F1,F3
"	6	0.079849	"	F1,F3
"	7	0.012796	"	F1
"	8	0.069982	"	F1,F2
"	9	0.012442	"	F1
"	10	0.001981	"	F1
"	11	0.013598	"	F1
"	12	0.001177	"	F1
"	13	0.020832	"	F1
"	14	0.102015	"	F1,F4
"	15	0.086270	"	F1,F2,F4

TABELLA 2.4

Si riferisce al segnale sintetico contenente la prima e la quinta frequenza.

SEGNALE	INDICE ESPERIM.	EMS - TEST	FREQUENZE TARGET	FREQUENZE OUTPUT
sintetico	1	0.020477	F1,F5	F1,F5
"	2	0.097939	"	F1,F3,F5
"	3	0.000274	"	F1,F5
"	4	0.061793	"	F1,F4,F5
"	5	0.059962	"	F1,F4,F5
"	6	0.060790	"	F1,F4,F5
"	7	0.007769	"	F1,F5
"	8	0.119966	"	F1,F4,F5
"	9	0.017626	"	F1,F5
"	10	0.001934	"	F1,F5
"	11	0.006179	"	F1,F5
"	12	0.002482	"	F1,F5
"	13	0.006518	"	F1,F5
"	14	0.000563	"	F1,F5
"	15	0.000037	"	F1,F5

TABELLA 2.5

Si riferisce al segnale sintetico contenente le frequenze 2, 4 e 5.

SEGNALE	INDICE ESPERIM.	EMS - TEST	FREQUENZE TARGET	FREQUENZE OUTPUT
sintetico	1	0.137113	F2,F4,F5	F2,F3,F4,F5
"	2	0.168187	"	F2,F3,F4,F5
"	3	0.009452	"	F2,F4,F5
"	4	0.004706	"	F2,F4,F5
"	5	0.003821	"	F2,F4,F5
"	6	0.004721	"	F2,F4,F5
"	7	0.010240	"	F2,F4,F5
"	8	0.065595	"	F2,F4,F5
"	9	0.006825	"	F2,F4,F5
"	10	0.036584	"	F2,F4,F5
"	11	0.010886	"	F2,F4,F5
"	12	0.048971	"	F2,F4,F5
"	13	0.020779	"	F2,F4,F5
"	14	0.078325	"	F2,F3,F4,F5
"	15	0.080978	"	F2,F3,F4,F5

TABELLA 2.6

Si riferisce al segnale sintetico contenente le frequenze 1, 2, 3 e 5.

SEGNALE	INDICE ESPERIM.	EMS - TEST	FREQUENZE TARGET	FREQUENZE OUTPUT
sintetico	1	0.020618	F1,F2,F3,F5	F1,F2,F3,F5
"	2	0.031333	"	F1,F2,F3,F5
"	3	0.008879	"	F1,F2,F3,F5
"	4	0.034308	"	F1,F2,F3,F5
"	5	0.034293	"	F1,F2,F3,F5
"	6	0.033304	"	F1,F2,F3,F5
"	7	0.046263	"	F1,F2,F3,F5
"	8	0.125645	"	F1,F3,F5
"	9	0.041305	"	F1,F2,F3,F5
"	10	0.070253	"	F1,F2,F3,F5
"	11	0.048273	"	F1,F2,F3,F5
"	12	0.072324	"	F1,F2,F3,F5
"	13	0.021677	"	F1,F2,F3,F5
"	14	0.068941	"	F1,F2,F3,F5
"	15	0.053070	"	F1,F2,F3,F5

TABELLE 3

Per completezza riportiamo adesso le tabelle relative ai valori output della rete per ciascuno dei 6 patterns di test. I dati numerici rappresentano la percentuale associata ad ogni frequenza. Un valore ≥ 50% indica la presenza della relativa frequenza nel segnale corrente.

TABELLA 3.1

SEGNALE	INDICE ESPER.	% FREQ. F1	% FREQ. F2	% FREQ. F3	% FREQ. F4	% FREQ. F5	% FREQ. F6
Tobenna	1	99.48	97.77	87.69	63.51	57.44	0.38
"	2	99.22	98.22	89.90	54.32	**27.69**	0.85
"	3	99.60	95.38	**49.98**	**38.19**	**42.65**	0.23
"	4	98.02	92.96	**48.22**	54.43	65.61	1.87
"	5	97.95	92.96	**48.22**	54.15	64.83	1.94
"	6	98.04	92.82	**48.22**	54.73	64.58	1.90
"	7	99.51	95.46	**48.03**	60.44	57.27	23.54
"	8	99.98	76.04	**49.56**	51.91	**49.14**	6.57
"	9	99.55	95.20	**49.03**	61.05	**49.76**	20.63
"	10	99.54	97.88	**43.48**	57.70	58.84	27.61
"	11	99.49	95.45	**40.84**	60.06	59.57	24.58
"	12	99.65	98.58	**43.79**	56.75	54.44	27.93
"	13	99.93	99.54	**46.02**	52.41	**22.46**	6.92
"	14	99.98	99.99	**47.16**	**49.65**	**17.43**	3.87
"	15	99.97	99.99	**45.83**	**49.91**	**16.22**	5.83

TABELLA 3.2 (segnale con tutte le sei frequenze)

SEGNALE	INDICE ESPER.	% FREQ. F1	% FREQ. F2	% FREQ. F3	% FREQ. F4	% FREQ. F5	% FREQ. F6
sintetico	1	91.24	98.91	81.29	99.84	98.67	99.39
"	2	99.86	99.95	**43.24**	99.97	99.99	100
"	3	99.88	99.61	**37.50**	92.45	96.27	99.73
"	4	81.30	99.75	82.40	97.29	91.17	93.90
"	5	79.60	99.63	80.23	97.28	91.00	93.71
"	6	81.03	99.74	82.11	97.19	90.99	94.08
"	7	98.96	98.52	99.22	99.09	99.71	99.22
"	8	99.99	93.78	**22.68**	99.99	98.00	99.85
"	9	99.01	98.29	99.02	98.39	99.64	98.83
"	10	98.56	99.18	99.52	99.88	99.79	99.57
"	11	98.91	98.60	99.25	99.24	99.71	99.29
"	12	98.78	99.38	99.59	99.90	99.81	99.55
"	13	99.86	99.79	95.51	99.71	97.53	99.87
"	14	99.99	99.99	99.76	99.99	99.88	100
"	15	99.99	99.99	99.70	99.99	99.94	99.99

TABELLA 3.3 (segnale con la prima frequenza)

SEGNALE	INDICE ESPER.	% FREQ. F1	% FREQ. F2	% FREQ. F3	% FREQ. F4	% FREQ. F5	% FREQ. F6
sintetico	1	99.98	4.33	35.87	3.34	0.05	20.39
"	2	99.66	**56.31**	**99.96**	1.11	0.01	0.0
"	3	99.96	**69.88**	**93.64**	**50.67**	0.0	12.04
"	4	99.99	1.87	**67.85**	19.63	0.38	1.18
"	5	99.99	2.10	**68.75**	23.13	0.36	1.28
"	6	99.99	1.94	**66.39**	19.41	0.40	1.15
"	7	99.70	24.98	11.84	1.41	0.82	0.42
"	8	99.98	**61.14**	0.0	21.43	0.20	0.85
"	9	99.75	25.48	9.40	2.62	0.90	0.63
"	10	99.44	10.27	3.54	0.13	0.51	0.10
"	11	99.65	25.34	13.07	1.10	0.84	0.35
"	12	99.41	7.84	2.91	0.09	0.42	0.08
"	13	99.90	32.27	1.19	3.06	0.20	14.03
"	14	99.89	38.39	0.0	**68.15**	0.0	0.21
"	15	99.82	**51.44**	0.0	**50.27**	0.0	0.93

TABELLA 3.4 (segnale contenente le frequenze 1 e 5)

SEGNA-LE	INDICE ESPER.	% FREQ. F1	% FREQ. F2	% FREQ. F3	% FREQ. F4	% FREQ. F5	% FREQ. F6
sintetico	1	99.99	0.32	16.81	23.69	99.94	7.12
"	2	99.94	0.0	**76.38**	1.64	94.08	0.09
"	3	99.99	1.90	0.93	0.0	99.94	3.37
"	4	99.99	0.35	29.28	**53.13**	99.97	1.87
"	5	99.99	0.47	28.76	**52.36**	99.97	2.03
"	6	99.99	0.35	29.17	**52.62**	99.97	1.98
"	7	99.91	3.08	13.79	12.56	97.24	7.60
"	8	99.99	0.22	0.32	**84.76**	99.99	2.81
"	9	99.90	4.16	26.68	15.34	96.93	3.93
"	10	99.94	1.02	6.03	3.37	97.27	5.86
"	11	99.91	2.83	9.58	11.49	97.30	9.73
"	12	99.95	0.87	7.62	2.40	97.62	6.33
"	13	99.80	1.85	10.36	16.02	99.94	1.55
"	14	99.93	0.0	5.59	0.01	99.99	0.72
"	15	99.92	0.0	0.32	0.38	99.99	1.30

TABELLA 3.5 (segnale contenente le frequenze 2, 4 e 5)

SEGNA-LE	INDICE ESPER.	% FREQ. F1	% FREQ. F2	% FREQ. F3	% FREQ. F4	% FREQ. F5	% FREQ. F6
sintetico	1	1.33	96.25	**90.23**	99.62	99.62	0.46
"	2	0.0	87.54	**94.28**	99.99	99.80	0.28
"	3	5.61	97.48	8.29	99.96	91.42	0.94
"	4	1.74	97.71	11.02	99.98	99.85	1.13
"	5	1.72	98.27	9.62	99.98	99.82	1.10
"	6	1.80	97.63	11.14	99.98	99.84	1.19
"	7	7.45	95.27	17.86	99.86	96.01	3.17
"	8	0.16	99.87	**60.62**	99.99	99.97	1.52
"	9	6.95	95.61	15.02	99.75	95.61	4.42
"	10	24.88	91.74	19.37	99.95	97.19	2.14
"	11	7.92	95.10	17.82	99.87	96.01	2.99
"	12	28.43	91.87	21.91	99.96	97.43	1.93
"	13	4.86	98.11	32.36	99.66	98.58	1.74
"	14	16.02	99.95	**51.33**	99.99	99.98	0.39
"	15	14.45	99.94	**57.86**	99.98	99.95	0.38

TABELLA 3.6 (segnale contenente le frequenze 1, 2, 3 e 5)

SEGNA-LE	INDICE ESPER.	% FREQ. F1	% FREQ. F2	% FREQ. F3	% FREQ. F4	% FREQ. F5	% FREQ. F6
sintetico	1	99.72	97.04	99.33	16.11	94.25	0.63
"	2	93.87	92.33	100	22.68	83.36	0.58
"	3	99.97	98.00	97.08	11.09	95.92	0.25
"	4	87.58	84.71	99.88	12.14	91.43	7.12
"	5	86.35	83.36	99.88	11.89	91.40	6.57
"	6	87.76	84.50	99.87	12.15	91.42	6.54
"	7	98.90	96.16	99.73	25.08	98.73	19.56
"	8	79.48	**46.98**	99.82	15.27	99.64	5.83
"	9	98.71	95.11	99.72	25.06	98.59	17.10
"	10	99.43	99.65	99.92	32.84	98.64	21.27
"	11	98.95	96.79	99.73	26.12	98.68	20.22
"	12	99.52	99.74	99.94	31.61	98.84	22.35
"	13	98.91	96.02	95.45	14.35	99.28	11.69
"	14	99.99	88.02	99.85	4.69	99.96	42.54
"	15	99.90	96.01	99.35	8.12	99.77	36.93

NOTA

I valori evidenziati in neretto corrispondono agli errori di valutazione delle reti in ogni esperimento.

5.4 ANALISI DEI RISULTATI SPERIMENTALI

Riportiamo nella prossima tabella un quadro riassuntivo dei comportamenti delle varie reti in fase di test. Utilizzeremo la seguente notazione :

segnale Tobenna reale contenente le prime 5 frequenze : TEST 1

segnale sintetico contenente tutte le frequenze : TEST 2

segnale sintetico contenente la prima frequenza : TEST 3

segnale sintetico contenente le frequenze F1 e F5 : TEST 4

segnale sintetico contenente le frequenze F2, F4 e F5 : TEST 5

segnale sintetico contenente le frequenze F1, F2, F3 e F5 : TEST 6

TABELLA 4

INDICE ESPER.	ERRORI TEST 1	ERRORI TEST 2	ERRORI TEST 3	ERRORI TEST 4	ERRORI TEST 5	ERRORI TEST 6	TOT. ERRORI
1	0	0	0	0	1	0	1
2	1	1	2	1	1	0	6
3	3	1	3	0	0	0	7
4	1	0	1	1	0	0	3
5	1	0	1	1	0	0	3
6	1	0	1	1	0	0	3
7	1	0	0	0	0	0	1
8	2	1	1	1	1	1	7
9	2	0	0	0	0	0	2
10	1	0	0	0	0	0	1
11	1	0	0	0	0	0	1
12	1	0	0	0	0	0	1
13	2	0	0	0	0	0	2
14	3	0	1	0	1	0	5
15	3	0	2	0	1	0	6

Dall'analisi della tabella 4, risulta una buona performance in generale di tutti i modelli utilizzati, tranne quelli in cui, indipendentemente dalla topologia, sono stati scelti valori piu' elevati per il tasso d'apprendimento eta e per il fattore momento alpha.

In particolare, i migliori modelli di rete, in termini di capacita' di generalizzazione, sono stati :

modello 12 - 6 - 6 - 6 con eta = 0.001 ed alpha = 0.1, (1 errore nel test 5);

modello 12 - 8 - 7 - 6 con eta = 0.002 ed alpha = 0.075, (1 errore nel test 1);

modello 12 - 8 - 7 - 6 con eta = 0.002 ed alpha = 0.1, (1 errore nel test 1);

modello 12 - 8 - 7 - 6 con eta = 0.001 ed alpha = 0.05, (1 errore nel test 1);

modello 12 - 8 - 7 - 6 con eta = 0.002 ed alpha = 0.05, (1 errore nel test 1);

Personalmente, riteniamo piu' affidabile il primo modello per due motivi :

1) e' l'unico modello ad avere totale assenza di errori nel test 1, corrispondente al segnale reale che, come tale e' sicuramente il test piu' significativo.

2) contiene il minor numero di neuroni negli strati interni, rispetto agli altri modelli dotati di una buona capacita' di generalizzazione, dunque e' migliore dal punto di vista della complessita' computazionale.

6 CONCLUSIONI

Riassumiamo in breve i contenuti salienti del presente lavoro di tesi:

Sono stati affrontati due problematiche fondamentali : la prima, connessa al tentativo di ricostruire le caratteristiche di una successione rocciosa nelle cosiddette interruzioni, cioe' laddove, per diverse fenomenologie naturali, non e' stato possibile rilevare le caratteristiche stratigrafiche mediante osservazioni dirette; la seconda, riguardante il riconoscimento nella successione di processi di stratificazione legati a particolari irregolarita' dell'orbita terrestre, (cicli di Milankovitch).

Per la risoluzione di tali problematiche sono stati utilizzati metodi estremamente originali, nel senso che per la prima volta abbiamo applicato dei modelli, per tale settore delle scienze della terra, finora utilizzati nel campo sismico e petrolifero, con un'elevata dose di rischio, nel senso che a priori eravamo confortati solo dalla realizzabilita' teorica di questi esperimenti, che ci ha incoraggiati e sostenuti durante tutta la durata del lavoro, soprattutto nei momenti di particolare panico!

I discreti successi ottenuti, dimostrano l'effettiva applicabilita' di metodologie neurali anche a questo settore di studi geologici, ampliando la categoria di problemi trattabili con esse.

Pertanto il presente lavoro di tesi e' da inquadrare nell'ottica di un'ulteriore spinta allo sviluppo di reti neurali ed una pionieristica applicazione di esse ad un'ulteriore branca della scienza.

APPENDICE A

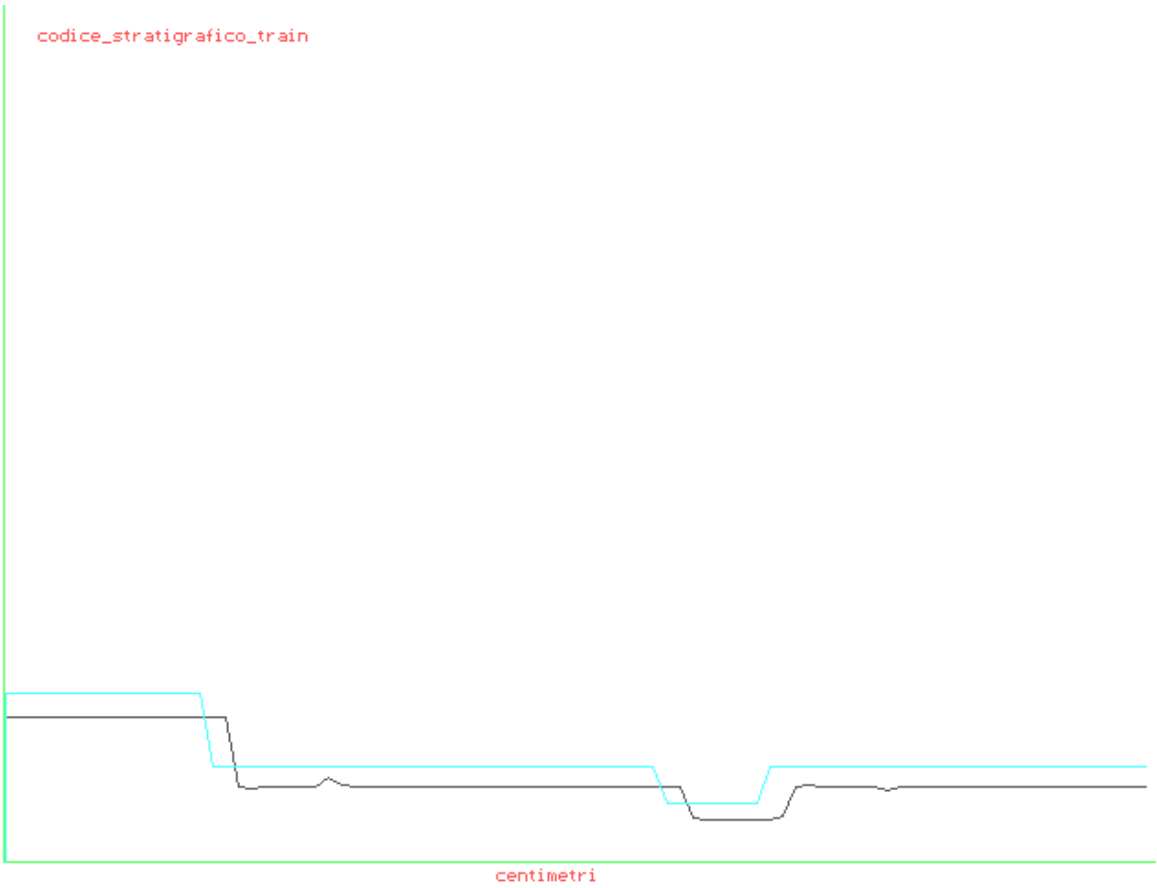
RISULTATI SPERIMENTALI RELATIVI ALLA RICOSTRUZIONE DI PORZIONI MANCANTI NEI SEGNALI STRATIGRAFICI.

ESPERIMENTO N. 1

Il primo esperimento riguarda l'uso di un numero piccolo di valori, variabili per la fase di training e costanti per la fase di test.

FASE DI TRAINING

Il grafico di seguito esposto si riferisce all'output della rete in fase di training:



La linea superiore si riferisce ai codici stratigrafici del segnale originale, mentre quella inferiore, a quelli appresi dalla rete.

I parametri relativi all'esperimento sono i seguenti :

NUMERO NEURONI INPUT	**10**
NUMERO NEURONI HIDDEN	**10**
NUMERO NEURONI OUTPUT	**1**
NUMERO PATTERNS	**89**
RANGE CENTIMETRI UTILIZZATI	**[311 , 399]**
ITERAZIONI	**20000**
LEARNING RATE	**0.01**
MOMENTUM	**0.6**
ERRORE MEDIO TOTALE	**0.002500**

FASE DI TEST

Il grafico di seguito riportato si riferisce all'output della rete in fase di test :

400 **439**

La linea superiore si riferisce ai codici del segnale originale, mentre quella inferiore a quelli ricostruiti dalla rete.

I parametri relativi alla fase di test sono i seguenti :

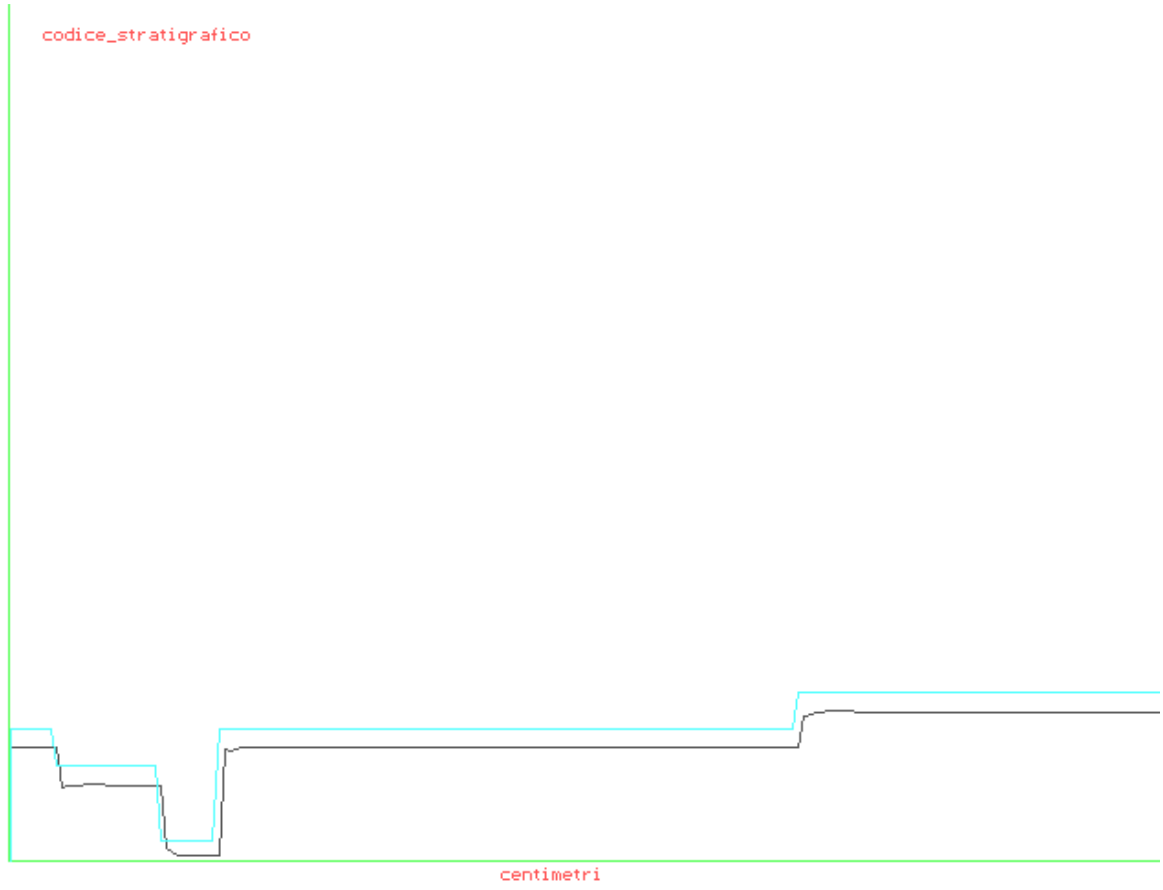
NUMERO DI NEURONI INPUT	**10**
NUMERO DI NEURONI HIDDEN	**10**
NUMERO DI NEURONI OUTPUT	**1**
NUMERO PATTERNS	**1**
RANGE CENTIMETRI DEL PATTERN	**[389 , 399]**
RANGE CENTIMETRI RICOSTRUITI	**[400 , 439]**
ERRORE MEDIO TOTALE	**0.050115**

ESPERIMENTO N. 2

Il secondo esperimento e' stato condotto su sequenze notevolmente piu' lunghe, in modo da testare il comportamento della rete anche in questo caso.

FASE DI TRAINING.

Il grafico visualizzato di seguito si riferisce all'output della rete dopo la fase di training, confrontato con la sequenza corrispondente del segnale originale :



11 **1999**

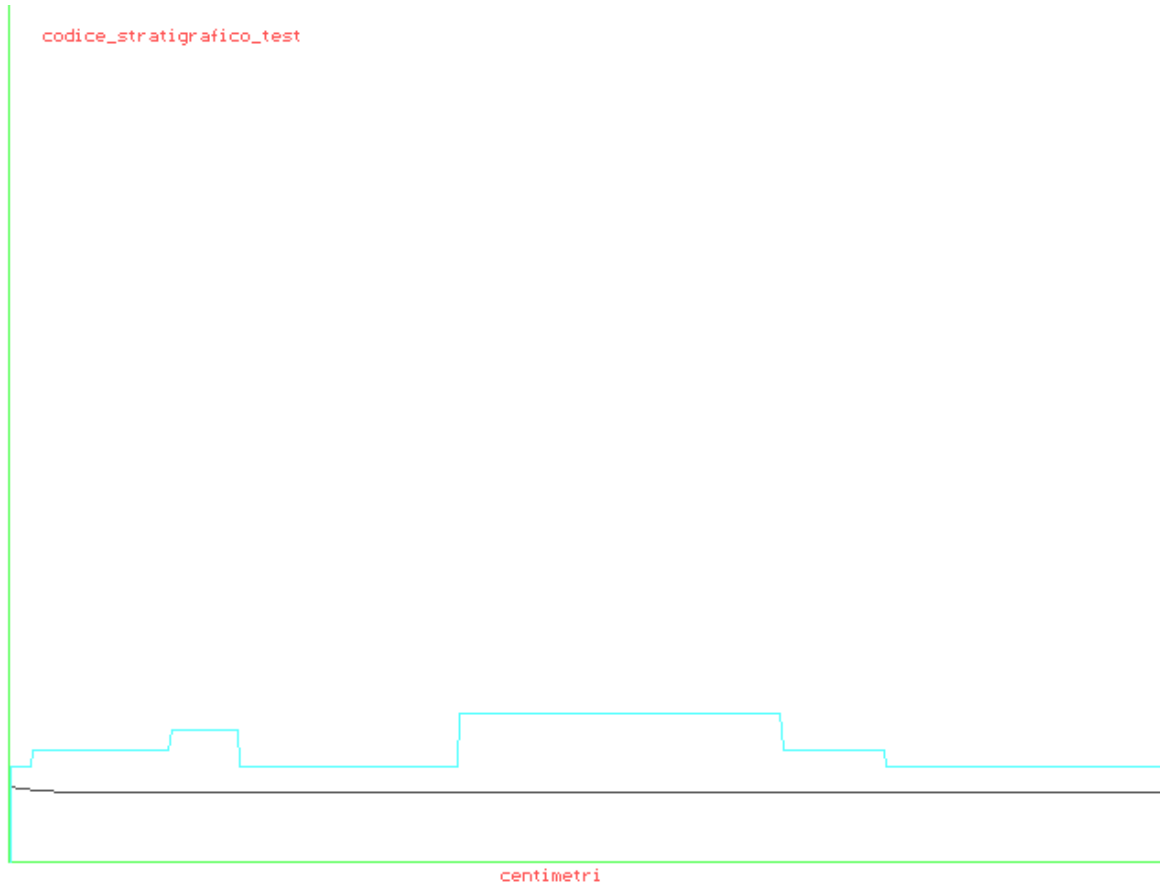
La linea superiore si riferisce ai codici stratigrafici del segnale originale, mentre quella inferiore a quelli appresi dalla rete.

I parametri relativi all'esperimento sono stati i seguenti :

NUMERO NEURONI INPUT	**10**
NUMERO NEURONI HIDDEN	**10**
NUMERO NEURONI OUTPUT	**1**
NUMERO PATTERNS	**1989**
RANGE CENTIMETRI UTILIZZATI	**[11 , 1999]**
ITERAZIONI	**17500**
LEARNING RATE	**0.01**
MOMENTUM	**0.6**
ERRORE MEDIO TOTALE	**0.001578**

FASE DI TEST

Il grafico di seguito riportato si riferisce al test eseguito dalla rete dopo la suddetta fase di addestramento :



2000 **2499**

La linea superiore rappresenta il segnale originale; quella inferiore, quello ricostruito.

I parametri relativi a questa fase dell'esperimento sono i seguenti :

NUMERO NEURONI INPUT	**10**
NUMERO NEURONI HIDDEN	**10**
NUMERO NEURONI OUTPUT	**1**
NUMERO PATTERNS	**1**
RANGE CENTIMETRI DEL PATTERN	**[1989 , 1999]**
RANGE CENTIMETRI RICOSTRUITI	**[2000 , 2499]**
ERRORE MEDIO TOTALE	**0.133691**

APPENDICE B

CODICE RELATIVO ALL'IMPLEMENTAZIONE DEL PERCETTRONE CON ALGORITMO DI BACK-PROPAGATION PER LA FASE TRAINING DELLA RICOSTRUZIONE DI INTERRUZIONI NEL SEGNALE STRATIGRAFICO.

```
/*        FILE DI COSTRUZIONE DEI PATTERNS PER PMS          */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>



void main()
{
   float *val;
   float temp;
   FILE *fd1,*fd2;
   int i,j,inizio;
   char filein[20],fileout[20];

   clrscr();

   val = (float far *) farcalloc(2000,sizeof(float));

   printf("\nnome file dati : ");
```

```
scanf("%s",filein);

if((fd1 = fopen(filein,"r")) == NULL)
 {
   fprintf(stderr,"\nimpossibile aprire il file %s",filein);
   exit(1);
 }

 printf("\nindice inizio sequenza (>= 1) : ");
 scanf("%d",&inizio);

 for(i=1;i < inizio;i++)
  fscanf(fd1,"%f\n",&temp);

 for(i=0;i < 2000;i++)
 {
  fscanf(fd1,"%f",&val[i]);
  printf("dato val[%d]=%f\r",i+inizio,val[i]);
 }
 fclose(fd1);

printf("\nnome file out : ");
scanf("%s",fileout);

fd2 = fopen(fileout,"w");
```

```
   for(i=0;i < 1990;i++)
    {
      printf("\ncostruzione pattern %d",i);
      for(j=i;j <= 9+i;j++)
       {
   fprintf(fd2,"%f ",val[j]);
       }
      fprintf(fd2,"%f %d\n",val[j],i);
    }

   fclose(fd2);
}



/*************************************************************************
*       PMS.C
*                MULTILAYER PERCEPTRON
*
*                    ALGORITMO DI BACK PROPAGATION
*
*          sorgente in c++ 3.0     (versione per MSDOS)
*                                                        *
*                                                      Max Brescia
*
*
*************************************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
#include <alloc.h>


#define   ESC          27
#define   ERRORLEVEL    0.02
#define   ITEMS        8
#define   MAXWEIGHT     ((float)0.3)
#define   SCALEWEIGHT   ((float)32767)


   /* tipi e prototipi per la memorizzazione dinamica degli array */

typedef float far *PFLOAT;
typedef PFLOAT VECTOR;
typedef PFLOAT *MATRIX;

void VectorAllocate(VECTOR *vector, int nCols);
void AllocateCols(PFLOAT matrix[], int nRows, int nCols);
```

```
void MatrixAllocate(MATRIX *pmatrix, int nRows, int nCols);
void MatrixFree(MATRIX matrix, int nRows);

/* def. memoria per i livelli della rete */

    /* array per input, output, pesi e target della rete */

MATRIX  out0;       /* input layer                          */
MATRIX  out1;       /* hidden1 layer                        */
MATRIX  delta1;     /* delta al livello hidden1             */
MATRIX  delw1;      /* cambio nei pesi input-hidden1        */
MATRIX  w1;         /* pesi input-hidden1                   */
MATRIX  out2;       /* livello output                       */
MATRIX  delta2;     /* delta al livello output              */
MATRIX  delw2;      /* cambio nei pesi hidden1-output       */
MATRIX  w2;         /* pesi hidden1-output                  */
MATRIX  out3;       /* 2ø livello hidden                    */
MATRIX  delta3;     /* delta secondo livello hidden         */
MATRIX  delw3;      /* cambio nei pesi hidden1-hidden2      */
MATRIX  w3;         /* pesi hidden1-hidden2                 */
MATRIX  target;     /* output target                        */
VECTOR PatternID;   /* identificatore per ogni pattern memorizzato */

int o;
```

```
float function(int f,float g)

{
  float valore;

  if (f == 1)

       valore = 1.0 / (1.0 + exp(-g));              /* sigmoide */



  if (f == 2)
      valore = tanh(g);                  /* tangente iperbolica */




   if (f == 3)

       valore = atan(g);                 /* arcotangente */


     return valore;
}
```

```
float diff_function(float x)
{
  float y;

  if (o == 1)
      y = x * (1.0 - x);

  if (o == 2)
      y = (1.0 / (cosh(x))) * (1.0 / (cosh(x)));

  if (o == 3)
      y = (1.0 / (1.0 + (x * x)));

  return(y);
}




void main(int argc, char *argv[])
{

 float eta = 0.15,              /* tasso di apprendimento per default */
       alpha = 0.075;           /* fattore momento per default        */

 int nReportErrors = 100;          /* frequenza di errore di riporto */
 float ErrorLevel  = ERRORLEVEL;   /* livello di errore              */
```

```
char MonitorError = 0;

float err;
float wmax,frand,valrand,seed;
float scale = SCALEWEIGHT;

float error;            /* ultimo valore errore della somma quadratica */
register int h;         /* indice livello hidden                       */
register int i;         /* indice livello input                        */
register int j;         /* indice livello output                       */
register int h3;        /* indice secondo livello hidden               */
int p,                  /* indice numero di pattern                    */
    q,                  /* indice numero di iterazione                 */
    r,                  /* indice numero di esecuzioni                 */
    nPatterns,          /* numero di patterns desiderati               */
    nInputNodes,        /* numero di nodi input                        */
    nHiddenNodes,       /* numero di nodi primo livello hidden         */
    nHiddenNodes3,      /* numero nodi secondo livello hidden          */
    nHiddenLayers,      /* numero livelli hidden                       */
    nOutputNodes,       /* numero di nodi output                       */
    nIterations,        /* numero di iterazioni desiderate             */
    label;              /* label                                       */

char tipo_features,
    tipo_target;

FILE *fpRun,        /* file configurazione */
```

```
    *fpcmd;         /* file comandi        */

char szResults[30];  /* vari nomi dei patterns */
char szError[30];
char szPattern[30];
char szWeights[30];
char szWeightsOut[30];

                    /* stringhe ausiliarie */
char string0[30];

char *progname = *argv;




/*----------------lettura di argomenti linea di comando-----------------*/

 clrscr();


 printf("\nArtificial Neural Network\n");
 printf("  Algoritmo di Back Propagation\n");

 for (; argc > 1; argc--)
```

```
  {
   char *arg = *++argv;
   if (*arg != '-')   break;
   switch (*++arg)
   {
     case 'e': sscanf(++arg, "%d", &nReportErrors); break;
     default:  break;
   }
  }


 if (argc < 2)
  {
fprintf(stderr,"\n    Uso: %s -en nome_file.cnf nome_file.cmd\n",progname);
   fprintf(stderr,"  -en => visualizzazione errore ogni n iterazioni\n");
   fprintf(stderr,"\nPer informazioni su formattazione files esterni\n");
   fprintf(stderr,"  vedere bp.doc\n");
   exit(1);
  }

 /*------------- Apertura file.cnf per lettura----------------------*/

 if ((fpRun = fopen(*argv, "r")) == NULL)

  {
    fprintf(stderr, "%s: impossibile aprire il file %s\n",progname,*argv);
    exit(1);
```

```
 }

/* lettura prima linea */

fscanf(fpRun,"%s %d", string0,&label);




/* ------------------inizio del ciclo di lavoro---------------------- */

  /* lettura e controllo */

  fscanf(fpRun,
"%s %d\n%s %c\n%s %d\n%s %c\n%s %d\n%s %d\n%s %d\n%s
%d\n\n\n%s %f\n%s %f\n%s %f\n%s %f\n%s %f",
  string0,
  &nInputNodes,        /* numero di nodi input */
  string0,
  &tipo_features,          /* tipo dati dei patterns */
  string0,
  &nOutputNodes,        /* numero di nodi output */
  string0,
  &tipo_target,        /* tipo dati target */
  string0,
```

```
    &nPatterns,          /* numero patterns input */
    string0,
    &nHiddenLayers,     /* numero livelli hidden */
    string0,
    &nHiddenNodes,       /* nø di neuroni primo livello hidden */
    string0,
    &nHiddenNodes3,   /* nø di neuroni secondo livello hidden */
    string0,
    &ErrorLevel,      /* tolleranza output */
    string0,
    &eta,             /* tasso di apprendimento */
    string0,
    &alpha,       /* fattore momento */
    string0,
    &seed,      /* valore random */
    string0,
    &valrand);   /* range valori random */


fclose(fpRun);


  /*------------allocazione di memoria dinamica per tutti i dati-------*/

  MatrixAllocate(&out0,       nPatterns,      nInputNodes);
  MatrixAllocate(&out1,       nPatterns,      nHiddenNodes);
  MatrixAllocate(&out2,       nPatterns,      nOutputNodes);
```

```
  MatrixAllocate(&delta2,     nPatterns,      nOutputNodes);
  MatrixAllocate(&delta1,     nPatterns,      nHiddenNodes + 1);
  MatrixAllocate(&delw1,      nHiddenNodes,   nInputNodes + 1);

  MatrixAllocate(&w1,         nHiddenNodes,   nInputNodes + 1);
if (nHiddenNodes3 != 0)
 {

  MatrixAllocate(&out3,       nPatterns,      nHiddenNodes3);
  MatrixAllocate(&delta3,     nPatterns,      nHiddenNodes3 + 1);
  MatrixAllocate(&delw3,      nHiddenNodes3,  nHiddenNodes + 1);
  MatrixAllocate(&w3,         nHiddenNodes3,  nHiddenNodes + 1);
  MatrixAllocate(&w2,         nOutputNodes,   nHiddenNodes3 + 1);
  MatrixAllocate(&delw2,      nOutputNodes,   nHiddenNodes3 + 1);
 }

else
   {
    MatrixAllocate(&w2,       nOutputNodes,   nHiddenNodes + 1);
    MatrixAllocate(&delw2,    nOutputNodes,   nHiddenNodes + 1);
   }

  MatrixAllocate(&target,     nPatterns,      nOutputNodes);
  VectorAllocate(&PatternID,  nPatterns);


/*-----------------lettura file.cmd---------------------------*/
```

```
if ((fpcmd = fopen(*++argv,"r")) == NULL)
 {
   fprintf(stderr,"\nimpossibile aprire il file %s\n",*argv);
   exit(1);
 }

 fscanf(fpcmd,"%s %s\n %s %s\n %s %d\n %s %s\n %s %s\n %s %s",
 string0,
 szWeightsOut,
 string0,
 szPattern,
 string0,
 &nIterations,
 string0,
 szResults,
 string0,
 szError,
 string0,
 szWeights);

 fclose(fpcmd);


/* generazione pesi iniziali RANDOM o contenuti in file esterno */
```

```
if ((wmax = valrand) == 0)
  wmax = MAXWEIGHT;

if (strcmp(szWeights,"RANDOM") == 0)
  {

  /* pesi random */

    srand(seed);


    for(h=0;h < nHiddenNodes;h++)
{
  for(i=0;i < nInputNodes+1;i++)
    {
      frand = rand();

      w1[h][i] = wmax * (1.0 - 2 * frand/scale);
      delw1[h][i] = 0.0;
    }
}


    if (nHiddenNodes3 != 0)
     {

 for(h3 = 0;h3 < nHiddenNodes3;h3++)
```

```
 {
   for(h = 0;h < nHiddenNodes+1;h++)
     {
         frand = rand();

         w3[h3][h] = wmax * (1.0 - 2 * frand/scale);
         delw3[h3][h] = 0.0;
     }
 }


 for(j = 0;j < nOutputNodes;j++)
  {
    for(h3 = 0;h3 < nHiddenNodes3+1;h3++)
      {
          frand = rand();
          w2[j][h3] = wmax * (1.0 - 2 * frand/scale);
          delw2[j][h3] = 0.0;
      }
  }
    }

  else
{

  for(j = 0;j < nOutputNodes;j++)
    {
```

```
      for(h = 0;h < nHiddenNodes+1;h++)
          {
            frand = rand();

            w2[j][h] = wmax * (1.0 - 2 * frand/scale);
            delw2[j][h] = 0.0;
          }
    }
}
  }


 else
  {

   /*----------lettura delle matrici dei pesi iniziali------------------*/

   if ((fpcmd = fopen(szWeights,"r")) == NULL)
    {
fprintf(stderr, "\n\n%s: impossibile aprire %s\n\n", progname, szWeights);
      exit(1);
    }


  /* lettura dei pesi input-hidden */
  for (h = 0; h < nHiddenNodes; h++)
    for(i = 0; i <= nInputNodes; i++)
```

```
  {
    fscanf(fpcmd, "%f", &w1[h][i]);


    delw1[h][i] = 0.0;
  }
if (nHiddenNodes3 != 0)
  {

   /* lettura pesi hidden1-hidden2 */

   for (h3 = 0; h3 < nHiddenNodes3; h3++)
     for(h = 0; h <= nHiddenNodes; h++)
       {
   fscanf(fpcmd,"%f",&w3[h3][h]);
   delw3[h3][h] = 0.0;
       }


   /* lettura pesi hidden2-output */

   for (j = 0; j < nOutputNodes; j++)
      for (h3 = 0; h3 <= nHiddenNodes3; h3++)
  {
   fscanf(fpcmd, "%f", &w2[j][h3]);
   delw2[j][h3] = 0.0;
  }
```

```
}    /* endif */

else
   {
     /* lettura pesi hidden1-output */

     for(j = 0; j < nOutputNodes; j++)
 for(h = 0; h <= nHiddenNodes; h++)
   {
     fscanf(fpcmd,"%f",&w2[j][h]);
     delw2[j][h] = 0.0;
   }
   }


     fclose(fpcmd);

}

/*----------lettura di tutti i patterns da imparare--------------------*/


 if ((fpcmd = fopen(szPattern, "r")) == NULL)
  {
 fprintf(stderr,"%s: impossibile aprire il file %s\n",progname,szPattern);
    exit(1);
```

```
  }

for (p = 0; p < nPatterns; p++)
  {
    for (i = 0; i < nInputNodes; i++)
   if (fscanf(fpcmd, "%f", &out0[p][i]) != 1)
   goto ALLPATTERNSREAD;

    /* lettura degli output target per lettura patterns input */

    for (j = 0; j < nOutputNodes; j++)
    fscanf(fpcmd, "%f", &target[p][j]);

    /* lettura degli identificatori di ogni pattern */

    fscanf(fpcmd, "%f", &PatternID[p]);

  }


ALLPATTERNSREAD:
fclose(fpcmd);


if (p < nPatterns)
 {
    fprintf(stderr,"%s: %d maggiori di %d patterns letti\n",
```

```
                                    progname,p,nPatterns);
    nPatterns = p;
  }


  /* apertura output file di errore */

  if ((fpcmd = fopen(szError, "w")) == NULL)
    {
      fprintf(stderr, "%s: impossibile aprire il file %s\n",
                                        progname,szError);


      exit(1);
    }
fprintf(stderr,nIterations > 1 ? "Sto provando...\n" : "Sto testando...\n");


  /*---------------- inizio loop iterazioni ----------------------------*/


  printf("\nINSERISCI IL TIPO FUNZIONE DI ATTIVAZIONE:\n");
  printf("\npossibili opzioni: \n1.logistica\n2.tanh\n3.arctan\n");
  scanf("%d",&o);


 printf("\n\nSe si desidera interrompere l'esecuzione digitare Ctrl-C");
```

```
printf("\n  Se si desidera riavviare l'apprendimento a partire\n");
printf("    dall'iterazione interrotta,inserire il nome del file\n");
printf("    temp.wei in %s come nuovo file pesi input\n\n",*argv);


 for (q = 0; q < nIterations; q++)
  {
    for (p = 0; p < nPatterns; p++)
     {

 /*--------------------livello hidden1-------------------------------*/

 /* somma input a livello hidden su tutte le combinazioni peso-input */

 for (h = 0; h < nHiddenNodes; h++)
   {
     float sum = w1[h][nInputNodes];           /*inizio col bias*/

     for (i = 0; i < nInputNodes; i++)
          sum += w1[h][i] * out0[p][i];

     out1[p][h] = function(o,sum);
   }



if (nHiddenNodes3 != 0)
```

```
  {


/*----------------------livello hidden2-----------------------------*/


for(h3 = 0; h3 < nHiddenNodes3; h3++)
  {
    float sum = w3[h3][nHiddenNodes];
    for(h = 0; h < nHiddenNodes; h++)
   sum += w3[h3][h] *out1[p][h];

    out3[p][h3] = function(o,sum);
  }


/*-------------------livello output-----------------------------*/

  for (j = 0; j < nOutputNodes; j++)
    {
       float sum = w2[j][nHiddenNodes3];
       for (h3 = 0; h3 < nHiddenNodes3; h3++)
                    sum += w2[j][h3] * out3[p][h3];

       out2[p][j] = function(o,sum);
    }
```

```
}    /* endif */

else
  {

  /*-----------------livello output--------------------------------*/

    for(j = 0; j < nOutputNodes; j++)
{
  float sum = w2[j][nHiddenNodes];
  for(h = 0; h < nHiddenNodes; h++)
         sum += w2[j][h] * out1[p][h];

  out2[p][j] = function(o,sum);
}
 }

/*-------------------delta output---------------------------------*/

   /* calcolo dei delta per ogni unita' output per un dato pattern */

 for (j = 0; j < nOutputNodes; j++)

 delta2[p][j] = (target[p][j]-out2[p][j]) * diff_function(out2[p][j]);
```

```
if (nHiddenNodes3 != 0)
 {


/*-------------------delta hidden2--------------------------------*/

 for(h3 = 0; h3 < nHiddenNodes3; h3++)
   {
     float sum = 0.0;
     for(j = 0; j < nOutputNodes; j++)
           sum += delta2[p][j] * w2[j][h3];

     delta3[p][h3] = sum * diff_function(out3[p][h3]);
   }


/*---------------delta hidden1-------------------------------*/

for(h = 0; h < nHiddenNodes; h++)
 {
   float sum = 0.0;
   for(h3 = 0; h3 < nHiddenNodes3; h3++)
    sum += delta3[p][h3] * w3[h3][h];

   delta1[p][h] = sum * diff_function(out1[p][h]);
 }
```

```
}   /* endif */

else
 {


  /*-------------------delta hidden1---------------------------*/

 for (h = 0; h < nHiddenNodes; h++)
  {
    float sum = 0.0;
    for (j = 0; j < nOutputNodes; j++)
          sum += delta2[p][j] * w2[j][h];

    delta1[p][h] = sum * diff_function(out1[p][h]);
  }
  }

}



if (nHiddenNodes3 != 0)
 {
```

```
/*----------------adattamento pesi hidden2-output--------------------*/

for(j = 0; j < nOutputNodes; j++)
 {
   float dw;                    /* peso delta */
   float sum = 0.0;

 /* somma dei delta per ogni nodo output per un'epoca */

 for(p = 0; p < nPatterns; p++)
   sum += delta2[p][j];



 /* calcolo nuovo peso bias per ogni unita' output */

 dw = eta * sum + alpha * delw2[j][nHiddenNodes3];
 w2[j][nHiddenNodes3] += dw;
 delw2[j][nHiddenNodes3] = dw;



 /* calcolo nuovi pesi */

 for(h3 = 0; h3 < nHiddenNodes3; h3++)
  {
```

```
      float sum = 0.0;
      for(p = 0; p < nPatterns; p++)
    sum += delta2[p][j] * out3[p][h3];

      dw = eta * sum + alpha * delw2[j][h3];
      w2[j][h3] += dw;
      delw2[j][h3] = dw;
     }
}


  /* adattamento pesi hidden2-hidden1 */

  for(h3 = 0; h3 < nHiddenNodes3; h3++)
    {
      float dw;
      float sum = 0.0;


/* somma dei delta per ogni nodo del secondo livello hidden per epoca */

      for(p = 0; p < nPatterns; p++)
    sum += delta3[p][h3];


/* calcolo nuovo peso bias per ogni unita' del secondo livello hidden */
```

```
    dw = eta * sum + alpha * delw3[h3][nHiddenNodes];
    w3[h3][nHiddenNodes] += dw;
    delw3[h3][nHiddenNodes] = dw;


      /* calcolo nuovi pesi */

    for(h = 0; h < nHiddenNodes; h++)
{
 float sum = 0.0;
 for(p = 0; p < nPatterns; p++)
     sum += delta3[p][h3] * out1[p][h];

 dw = eta * sum + alpha * delw3[h3][h];
 w3[h3][h] += dw;
 delw3[h3][h] = dw;
}
  }
}     /* endif */

 else
  {

   /*-------------adattamento dei pesi hidden1-output-------------*/

for (j = 0; j < nOutputNodes; j++)
 {
```

```
  float dw;                          /* peso delta */
  float sum = 0.0;


  /* somma dei delta per ogni nodo output per un'epoca */

  for (p = 0; p < nPatterns; p++)
sum += delta2[p][j];


  /* calcolo nuovo peso bias per ogni unita' output */

  dw = eta * sum + alpha * delw2[j][nHiddenNodes];
  w2[j][nHiddenNodes] += dw;
  delw2[j][nHiddenNodes] = dw;          /* delta per il bias */


  /* calcolo nuovi pesi */

  for (h = 0; h < nHiddenNodes; h++)
    {
 float sum = 0.0;
 for (p = 0; p < nPatterns; p++)
     sum += delta2[p][j] * out1[p][h];

 dw = eta * sum + alpha * delw2[j][h];
 w2[j][h] += dw;
```

```
  delw2[j][h] = dw;
      }
  }
}      /* end else */



  /*-----------adattamento pesi input-hidden1-------------------------*/

  for (h = 0; h < nHiddenNodes; h++)
    {


      float dw;
      float sum = 0.0;
      for (p=0; p < nPatterns; p++)
   sum += delta1[p][h];

      /* calcolo nuovo peso bias per ogni unita' hidden */

      dw = eta * sum + alpha * delw1[h][nInputNodes];
      w1[h][nInputNodes] += dw;
      delw1[h][nInputNodes] = dw;


      /* calcolo nuovi pesi */
```

```
    for (i = 0; i < nInputNodes; i++)
 {
  float sum = 0.0;
  for (p = 0; p < nPatterns; p++)
          sum += delta1[p][h] * out0[p][i];

  dw = eta * sum + alpha * delw1[h][i];
  w1[h][i] += dw;
  delw1[h][i] = dw;
 }
  }

/*----------------controllo tastiera---------------------------------*/

if (kbhit())
 {
    int c = getchar();
    if ((c = toupper(c)) == 'E')
  MonitorError++;
  else if (c == ESC)
      break;
  }

 /*-----------------errore di somma quadratica-------------------------*/

fprintf(fpRun,"\nVALORI ERRORE SU TUTTI I PATTERNS\n\n\n");
```

```
 if (MonitorError || (q % nReportErrors == 0))
   {



     for (p = 0, error = 0.0; p < nPatterns; p++)
  {
    for (j = 0;j < nOutputNodes; j++)
      {
        float temp = target[p][j] - out2[p][j];
        error += temp * temp;
      }
  }
/* errore medio per nodo su tutti i patterns */

  error /= (nPatterns * nOutputNodes);



  /* stampa il numero d'iterazione ed il valore errore */

fprintf(stderr,"Iterazione %5d/%-5d   Errore %f\r",q,nIterations,error);

  MonitorError = 0;



  if (q % nReportErrors == 0)
```

```
    fprintf(fpcmd,"\niterazione %d    errore : %f\n\n",q,error);



/*------------------stampa pesi finali---------------------------------*/

if ((fpRun = fopen(szWeightsOut,"w")) == NULL)

   {
fprintf(stderr, "%s: impossibile scrivere il file %s\n",
            progname,szWeightsOut);
     exit(1);
   }

fprintf(fpRun,"\n  STATO DELLA RETE ALL'ITERAZIONE %d\n",q);
fprintf(fpRun,"\n\n  sono compresi i neuroni bias!\n\n\n");

fprintf(fpRun,"  PESI INPUT-HIDDEN1 :\n\n");

for (h = 0; h < nHiddenNodes; h++)
   for (i = 0; i <= nInputNodes; i++)
fprintf(fpRun,"   in_hid1[%d][%d] = %g\n\n",i,h,w1[h][i]);



if (nHiddenNodes3 != 0)
```

```
{

  fprintf(fpRun,"  PESI HIDDEN1-HIDDEN2 :\n\n");

  for(h3 = 0; h3 < nHiddenNodes3; h3++)
   for(h = 0; h < nHiddenNodes+1; h++)
fprintf(fpRun,"   hid1_hid2[%d][%d] = %g\n\n",h,h3,w3[h3][h]);

  fprintf(fpRun,"  PESI HIDDEN2-OUTPUT :\n\n");

  for (j = 0;j < nOutputNodes;j++)
    for (h3 = 0;h3 < nHiddenNodes3+1;h3++)
 fprintf(fpRun,"   hid2_out[%d][%d] = %g\n\n",h3,j,w2[j][h3]);


}

 else
 {
  fprintf(fpRun,"  PESI HIDDEN1-OUTPUT :\n\n");

   for (j = 0; j < nOutputNodes; j++)
    for (h = 0; h <= nHiddenNodes; h++)
 fprintf(fpRun,"   hid1_out[%d][%d] = %g\n\n",h,j,w2[j][h]);

 }
```

```
 fclose(fpRun);


if ((fpRun = fopen("temp.wei","w")) == NULL)
  {
    fprintf(stderr,"\nimpossibile aprire il file\n");
    exit(1);
  }

    for(h = 0;h < nHiddenNodes;h++)
      for(i = 0;i < nInputNodes+1;i++)
  fprintf(fpRun,"%9.6f\n",w1[h][i]);

   if (nHiddenNodes3 != 0)
    {



      for(h3 = 0;h3 < nHiddenNodes3;h3++)
  for(h = 0;h < nHiddenNodes+1;h++)
    fprintf(fpRun,"%9.6f\n",w3[h3][h]);

      for(j = 0;j < nOutputNodes;j++)
  for(h3 = 0;h3 < nHiddenNodes3+1;h3++)
    fprintf(fpRun,"%9.6f\n",w2[j][h3]);
    }
```

```
   else
     {
  for(j = 0;j < nOutputNodes;j++)
    for(h = 0;h < nHiddenNodes+1;h++)
      fprintf(fpRun,"%9.6f\n",w2[j][h]);
      }


   fclose(fpRun);



/*------------------stampa valori attivazione finale---------------------*/


if ((fpRun = fopen(szResults,"w")) == NULL)
  {
    fprintf(stderr, "%s: impossibile scrivere il file %s\n",
                                        progname,szResults);
    fpRun = stderr;
  }
```

```
/* stampa vettore output finale */

fprintf(fpRun,"\n   OUTPUT DELLA RETE ALL'ITERAZIONE %d\n\n\n",q);
fprintf(fpRun,"parametri usati :\n");
fprintf(fpRun,"\n     eta   = %f\n",eta);
fprintf(fpRun,"       alpha = %f\n\n\n",alpha);

for (p = 0; p < nPatterns; p++)
  {
    for (j = 0; j < nOutputNodes; j++)
     {
 err = target[p][j] - out2[p][j];
 if (err < 0)  err = err * -1;

 fprintf(fpRun, "  out[%d] = %f",j,out2[p][j]);
     }
    fprintf(fpRun,"  pattern %-6.0f  errore = %f\n",PatternID[p],err);
  }

fclose(fpRun);



 if (error < ErrorLevel)
    break;
}
```

```
 }       /* end ciclo for delle iterazioni */



/*---------------fine del loop d'iterazione-----------------------------*/

for (p = 0, error = 0.0; p < nPatterns; p++)
  {
    for (j = 0; j < nOutputNodes; j++)
      {
  float temp = target[p][j] - out2[p][j];
  error += temp * temp;
      }
  }


  /* errore medio su tutti i patterns */


  error /= (nPatterns * nOutputNodes);


  /* stampa numero d'iterazione finale e valore di errore */

fprintf(stderr,"Iterazione %5d/%-5d   Errore %f\n",q,nIterations,error);
fprintf(fpcmd, "\niterazione finale %d       errore : %f\n\n",q,error);
```

```
   fclose(fpcmd);



/*--------------memoria dinamica libera per dati-------------------------*/

MatrixFree(out0,          nPatterns);
MatrixFree(out1,          nPatterns);
MatrixFree(delta1,        nPatterns);
MatrixFree(delw1,         nHiddenNodes);
MatrixFree(w1,            nHiddenNodes);
MatrixFree(out2,          nPatterns);
MatrixFree(delta2,        nPatterns);
MatrixFree(delw2,         nOutputNodes);
MatrixFree(w2,            nOutputNodes);

if (nHiddenNodes3 != 0)
{

 MatrixFree(out3,          nPatterns);
 MatrixFree(delta3,        nPatterns);
 MatrixFree(delw3,         nHiddenNodes3);
 MatrixFree(w3,            nHiddenNodes3);

}

MatrixFree(target,        nPatterns);
```

```
free(PatternID);


printf ("\n PER I RISULTATI CONSULTARE I SEGUENTI FILES :\n");
printf("  %s  per valori output\n",szResults);
printf("  %s  per pesi finali\n",szWeightsOut);
printf("  %s  per errori\n",szError);

}

/*-------------routines di allocazione di memoria per array--------------*/

/*spazio allocato per vettore di celle float per un vector[cols] dinamico*/

void VectorAllocate(VECTOR *vector, int nCols)
{
  if ((*vector = (VECTOR) farcalloc(nCols, sizeof(float))) == NULL)
    {
      fprintf(stderr, "VectorAllocate memoria insufficiente per i nodi\n");
      exit(1);
    }
}

/* spazio allocato per colonne (celle float) per una matrice dinamica */

void AllocateCols(PFLOAT matrix[], int nRows, int nCols)
{
```

```
  int i;
  for (i = 0; i < nRows; i++)
     VectorAllocate(&matrix[i], nCols);
}
/* spazio allocato per una matrice dinamica */

void MatrixAllocate(MATRIX *pmatrix, int nRows, int nCols)
{
  if ((*pmatrix = (MATRIX) farcalloc(nRows, sizeof(PFLOAT))) == NULL)
    {
       fprintf(stderr,"MatrixAllocate memoria insufficiente per i nodi\n");
       exit(1);
    }

  AllocateCols(*pmatrix,nRows,nCols);

 }


/* spazio libero per un array bidimensionale dinamico */

void MatrixFree(MATRIX matrix,int nRows)
  {
    int i;
    for (i = 0; i < nRows; i++)
       farfree(matrix[i]);
```

```
    farfree(matrix);
}
```

APPENDICE C

CODICE RELATIVO ALL'IMPLEMENTAZIONE DEL PERCETTRONE PER FASE DI TEST PER LA RICOSTRUZIONE DI INTERRUZIONI NEL SEGNALE STRATIGRAFICO.

```
/*************************************************************************
*       TESTNET.C
*               MULTILAYER PERCEPTRON
*
*                 ALGORITMO DI BACK PROPAGATION
*
*         sorgente in c++ 3.0     (versione per MSDOS)
*
*
*                                                   Max Brescia
*
*************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
#include <alloc.h>
```

```
   /* tipi e prototipi per la memorizzazione dinamica degli array */

typedef float far *PFLOAT;
typedef PFLOAT VECTOR;
typedef PFLOAT *MATRIX;

void VectorAllocate(VECTOR *vector, int nCols);
void AllocateCols(PFLOAT matrix[], int nRows, int nCols);
void MatrixAllocate(MATRIX *pmatrix, int nRows, int nCols);
void MatrixFree(MATRIX matrix, int nRows);

/* def. memoria per i livelli della rete */

     /* array per input, output, pesi e target della rete */


MATRIX  out0;       /* input layer                      */
MATRIX  out1;       /* hidden1 layer                      */
MATRIX  w1;         /* pesi input-hidden1                   */
MATRIX  out2;       /* livello output                   */
MATRIX  w2;         /* pesi hidden1-output                  */
VECTOR PatternID;    /* identificatore per ogni pattern memorizzato */

int o;
```

```
float function(int f,float g)

{
  float valore;

  if (f == 1)

       valore = 1.0 / (1.0 + exp(-g));              /* sigmoide */


  if (f == 2)
      valore = tanh(g);                  /* tangente iperbolica */



   if (f == 3)

       valore = atan(g);                  /* arcotangente */


     return valore;
}

void main(int argc, char *argv[])
```

```
{

 int nReportErrors = 1;
 register int h;          /* indice livello hidden                    */
 register int i;          /* indice livello input                     */
 register int j;          /* indice livello output                    */
 int p = 0;               /* indice numero di pattern                 */
 int newp,



     q,                   /* indice numero di iterazione              */
     r,                   /* indice numero di esecuzioni              */
     nPatterns,           /* numero di patterns desiderati            */
     nInputNodes,         /* numero di nodi input                     */
     nHiddenNodes,        /* numero di nodi primo livello hidden      */
     nOutputNodes,        /* numero di nodi output                    */
     nIterations,         /* numero di iterazioni desiderate          */
     label;               /* label                                    */

     float diff,percent;

char tipo_features,
     tipo_target;

 FILE *fpRun,         /* file configurazione */
      *fpcmd;         /* file comandi        */
```

```
char szResults[30];  /* vari nomi dei patterns */
char szPattern[30];
char szWeights[30];

                    /* stringhe ausiliarie */
char string0[30];

char *progname = *argv;

/*----------------lettura di argomenti linea di comando-----------------*/

 clrscr();

 printf("\nArtificial Neural Network\n");
 printf("  Algoritmo di Back Propagation\n");

 for (; argc > 1; argc--)
   {
    char *arg = *++argv;
    if (*arg != '-')   break;
    switch (*++arg)
    {
      case 'e': sscanf(++arg, "%d", &nReportErrors); break;
      default:  break;
    }
   }
```

```
 if (argc < 2)
   {
fprintf(stderr,"\n    Uso: %s -en nome_file.cnf nome_file.cmd\n",progname);
    fprintf(stderr,"  -en => visualizzazione errore ogni n iterazioni\n");
    fprintf(stderr,"\nPer informazioni su formattazione files esterni\n");
    fprintf(stderr,"  vedere bp.doc\n");
    exit(1);
   }

  /*------------- Apertura file.cnf per lettura----------------------*/

  if ((fpRun = fopen(*argv, "r")) == NULL)

  {
    fprintf(stderr, "%s: impossibile aprire il file %s\n",progname,*argv);
    exit(1);
  }

  /* lettura prima linea */

  fscanf(fpRun,"%s %d", string0,&label);


  /* -------------------inizio del ciclo di lavoro--------------------- */

    /* lettura e controllo */
```

```
      fscanf(fpRun,
"%s %d\n%s %c\n%s %d\n%s %c\n%s %d\n%s %d",
      string0,
      &nInputNodes,        /* numero di nodi input */
      string0,
      &tipo_features,          /* tipo dati dei patterns */
      string0,
      &nOutputNodes,        /* numero di nodi output */
      string0,
      &tipo_target,        /* tipo dati target */
      string0,
      &nPatterns,          /* numero patterns input */
      string0,
      &nHiddenNodes);       /* nø di neuroni primo livello hidden */


  fclose(fpRun);

    /*------------allocazione di memoria dinamica per tutti i dati-------*/

    MatrixAllocate(&out0,       nPatterns,      nInputNodes);
    MatrixAllocate(&out1,       nPatterns,      nHiddenNodes);
    MatrixAllocate(&out2,       nPatterns,      nOutputNodes);
    MatrixAllocate(&w1,         nHiddenNodes,   nInputNodes + 1);
    MatrixAllocate(&w2,       nOutputNodes,   nHiddenNodes + 1);
    VectorAllocate(&PatternID,  nPatterns);
```

```
/*-----------------lettura file.cmd----------------------------*/

if ((fpcmd = fopen(*++argv,"r")) == NULL)
 {
   fprintf(stderr,"\nimpossibile aprire il file %s\n",*argv);
   exit(1);
 }

 fscanf(fpcmd,"%s %s\n %s %d\n %s %s\n %s %s",
 string0,
 szPattern,
 string0,
 &nIterations,
 string0,
 szResults,
 string0,
 szWeights);

 fclose(fpcmd);


  /*----------lettura delle matrici dei pesi iniziali-----------------*/

  if ((fpcmd = fopen(szWeights,"r")) == NULL)
   {
fprintf(stderr, "\n\n%s: impossibile aprire %s\n\n", progname, szWeights);
```

```
    exit(1);
  }


 /* lettura dei pesi input-hidden */
 for (h = 0; h < nHiddenNodes; h++)
    for(i = 0; i <= nInputNodes; i++)
{
  fscanf(fpcmd, "%f", &w1[h][i]);
}

    /* lettura pesi hidden1-output */

    for(j = 0; j < nOutputNodes; j++)
 for(h = 0; h <= nHiddenNodes; h++)
   {
    fscanf(fpcmd,"%f",&w2[j][h]);
   }


    fclose(fpcmd);

/*---------- lettura del primo pattern da testare --------------------*/

if ((fpcmd = fopen(szPattern, "r")) == NULL)
  {
fprintf(stderr,"%s: impossibile aprire il file %s\n",progname,szPattern);
```

```
    exit(1);
   }

     for (i = 0; i < nInputNodes; i++)
    if (fscanf(fpcmd, "%f", &out0[p][i]) != 1)
    goto ALLPATTERNSREAD;

     /* lettura dell'identificatore del pattern */

     fscanf(fpcmd, "%f", &PatternID[p]);

  ALLPATTERNSREAD:
  fclose(fpcmd);


fprintf(stderr,nIterations > 1 ? "Sto provando...\n" : "Sto testando...\n");

   /*---------------- inizio loop iterazioni ----------------------------*/

   printf("\nINSERISCI IL TIPO FUNZIONE DI ATTIVAZIONE:\n");
   printf("\npossibili opzioni: \n1.logistica\n2.tanh\n3.arctan\n");
   scanf("%d",&o);

if ((fpRun = fopen(szResults,"w")) == NULL)
  {
    fprintf(stderr, "%s: impossibile scrivere il file %s\n",
                                        progname,szResults);
```

```
  exit(1);
}

fprintf(fpRun,"        %s\n\nneuroni hidden %d\nnumero patterns %d\n\n",
                    szResults,nHiddenNodes,nPatterns);


percent = 0.0;

 for (q = 0; q < nIterations; q++)
  {
    p = 0;
    while(p < nPatterns)
     {

 /*--------------------livello hidden1-------------------------------*/

 /* somma input a livello hidden su tutte le combinazioni peso-input */

 for (h = 0; h < nHiddenNodes; h++)
   {
     float sum = w1[h][nInputNodes];           /*inizio col bias*/

     for (i = 0; i < nInputNodes; i++)
          sum += w1[h][i] * out0[p][i];

     out1[p][h] = function(o,sum);
```

```
    }



    /*-----------------livello output---------------------------------*/

      for(j = 0; j < nOutputNodes; j++)
  {
    float sum = w2[j][nHiddenNodes];
    for(h = 0; h < nHiddenNodes; h++)
          sum += w2[j][h] * out1[p][h];

    out2[p][j] = function(o,sum);
  }


/*------------------stampa valori attivazione finale----------------------*/


 /* stampa vettore output finale */

     for (j = 0; j < nOutputNodes; j++)
      {
  fprintf(fpRun, "  pattern %d -- out[%d] = %f\n",p,j,out2[p][j]);
      }

/*--------------- costruzione patterns da testare --------------------*/
```

```
if(p <= 9)
{
  diff = 0.4 - out2[p][0];
  percent += diff;
  newp = p + 1;
}

else
 {
   diff = 0.5 - out2[p][0];
   percent += diff;
   newp = p + 1;
 }

for(i=0;i < nInputNodes-1;i++)
     out0[newp][i] = out0[p][i+1];
out0[newp][i] = out2[p][0];

p++;

}  /* end ciclo for per i patterns */

fprintf(fpRun,"\npercentuale media d'errore totale su tutti i pattern : %f",
            fabs(percent/nPatterns));

fclose(fpRun);
```

```
}   /* end ciclo for per l'iterazione */

printf("\npercentuale media d'errore totale : %f",fabs(percent/nPatterns));

/*--------------memoria dinamica libera per dati-------------------------*/

MatrixFree(out0,         nPatterns);
MatrixFree(out1,         nPatterns);
MatrixFree(w1,           nHiddenNodes);
MatrixFree(out2,         nPatterns);
MatrixFree(w2,           nOutputNodes);

free(PatternID);


printf ("\n PER I RISULTATI CONSULTARE I SEGUENTI FILES :\n");
printf("  %s  per valori output\n",szResults);

}

/*-------------routines di allocazione di memoria per array--------------*/

/*spazio allocato per vettore di celle float per un vector[cols] dinamico*/

void VectorAllocate(VECTOR *vector, int nCols)
{
  if ((*vector = (VECTOR) farcalloc(nCols, sizeof(float))) == NULL)
```

```
    {
      fprintf(stderr, "VectorAllocate memoria insufficiente per i nodi\n");
      exit(1);
    }
}
/* spazio allocato per colonne (celle float) per una matrice dinamica */

void AllocateCols(PFLOAT matrix[], int nRows, int nCols)
{
  int i;
  for (i = 0; i < nRows; i++)
     VectorAllocate(&matrix[i], nCols);
}

/* spazio allocato per una matrice dinamica */

void MatrixAllocate(MATRIX *pmatrix, int nRows, int nCols)
{
  if ((*pmatrix = (MATRIX) farcalloc(nRows, sizeof(PFLOAT))) == NULL)
    {
       fprintf(stderr,"MatrixAllocate memoria insufficiente per i nodi\n");
       exit(1);
    }

  AllocateCols(*pmatrix,nRows,nCols);

 }
```

```
/* spazio libero per un array bidimensionale dinamico */

void MatrixFree(MATRIX matrix,int nRows)
  {
    int i;
    for (i = 0; i < nRows; i++)
       farfree(matrix[i]);

    farfree(matrix);
}
```

APPENDICE D

CODICE RELATIVO ALL'IMPLEMENTAZIONE DEL METODO DI ANALISI SPETTRALE "PERIODOGRAMMA".

```
/*****************************************************************************
**
*
*            SORGENTE DI PGRAMMMA.C
*
*      PROGRAMMA DI CALCOLO PERIODOGRAMMA DI SERIE SPAZIALI
*      DI DATI EQUAMENTE SPAZIATI.
*
*       VERSIONE PER MS-DOS
*
*                    Per eseguire : pgramma.exe
*
*
*                                                  Max Brescia
*****************************************************************************
*/


#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <graphics.h>
```

```
#include <math.h>
#include <conio.h>
#include <dos.h>
#include <alloc.h>
#include <string.h>


#define  FREQ0   900
#define  FREQ1   200
#define  TIME    90
#define CLIP_ON 1   /* activates clipping in viewport */


void intestazione();
int display_out();
int display_prosp();
void menu(void);
int prog();
void display_graph();
void bordo(int startx,int starty,int endx,int endy);

double zo=0.0;


void main()
{
  clrscr();
```

```
  intestazione();

  sound(FREQ0);
  delay(TIME);          /* suona per 1/2 secondo */
  nosound();

  menu();
 }


 void intestazione()
 {
  clrscr();
  window(1,1,80,25);
  textcolor(2);

  gotoxy(2,2);
  cprintf("
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++");
  gotoxy(2,3);
  cprintf("            + CALCOLO PERIODOGRAMMA DI SERIE SPAZIALI DI
DATI +");
  gotoxy(2,4);
  cprintf("
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++");
 }
```

```
int prog()
{
  int num_dati,i,j,scelta,choice,k,m;
  float freq,sum;
  FILE *fp;
  char *c;
  float perc,tot,TOT,media1,varianza1,media,varianza,W;
  double esp,sum4,sum6,fraz1,fraz2,pgramma,p_normal;
  float num,po,esp1,esp2;
  double sum1,sum2,sum3,sum5,tau,rap;
  float far *ome;


  float far *X;
  float far *Y;
  char file_dati[20];
  int NI;
  float far *out;
  char file_out[20];
  char file_prosp[20];
  int first,second,third;
  float dato,flo_second;
  double pi = 3.141592654;
```

```
/*------------------------finestra 1---------------------------------*/


  sound(FREQ0);
  delay(TIME);
  nosound();

  clrscr();
  textcolor(3);
  bordo(3,6,50,9);
  textcolor(5);
  gotoxy(2,2);
  cprintf("nome file dati ? ");
  cscanf("%s",file_dati);
  clrscr();


  if((fp = fopen(file_dati,"r")) == NULL)
   {
     sound(FREQ1);
     delay(TIME);
     nosound();
     clrscr();
     textcolor(4+128);
     gotoxy(8,10);
     cprintf("impossibile aprire il file %s",file_dati);
     *c = getchar();
```

```
   return 1;
 }

sound(FREQ0);
delay(TIME);
nosound();


bordo(3,6,57,9);
textcolor(3);
gotoxy(2,2);
cprintf("numero dati campione da esaminare ? ");
cscanf("%d",&num_dati);
clrscr();


sound(FREQ0);
delay(TIME);
nosound();

bordo(3,6,57,9);
textcolor(4);
gotoxy(2,2);

cprintf("intervallo di smoothing (DISPARI) ? ");
cscanf("%d",&m);
clrscr();
```

```
if((Y = (float far *) farcalloc(num_dati,sizeof(float))) == NULL)
  {
    clrscr();
    sound(FREQ1);
    delay(TIME);
    nosound();
    textcolor(4+128);
    gotoxy(8,10);
    cprintf("memoria esaurita per array dei dati input");
    *c = getchar();
    return 1;
  }

if((X = (float far *) farcalloc(num_dati-m+1,sizeof(float))) == NULL)
  {
    clrscr();
    sound(FREQ1);
    delay(TIME);
    nosound();
    textcolor(4+128);
    gotoxy(8,10);
    cprintf("memoria esaurita per array dei dati input");
    *c = getchar();
    return 1;
  }
```

```
 i = 0;

 while(i < num_dati)
  {
    i++;
    fscanf(fp,"%f",&Y[i]);
  }

 fclose(fp);


/*---------- calcolo media e varianza dei dati originali ---------------- */

       tot = 0.0;
       TOT = 0.0;

       i = 0;
       while(i < num_dati)
        {
            i++;
            tot += Y[i];
            TOT += Y[i] * Y[i];
        }

   media1 = tot / (float)num_dati;
varianza1 = (TOT - ((tot * tot) / (float)num_dati)) / (float)(num_dati - 1);
```

```
/*--------------------- smoothing dei dati iniziali ----------------------*/


   if((fp = fopen("dati.out","w")) == NULL)
   {
      sound(FREQ1);
     delay(TIME);
     nosound();
     clrscr();
     textcolor(4+128);
     gotoxy(8,10);
     cprintf("impossibile aprire il file %s","dati.out");
     *c = getchar();
     return 1;

   }


 i = 0;
 while(i < (num_dati-m+1))
  {
   i++;

   sum = 0.0;
```

```
j = 0;
while(j < m)
 {
  j++;

  k = i+j-1;

  sum += Y[k];
 }

dato = sum / (float)m;
first = (int)dato;
dato = dato - first;
dato = dato * 10;
second = (int)dato;
dato = dato - second;
dato = dato * 10;
third = (int)dato;
if(third > 5) second = second + 1;

fprintf(fp,"%d.%d\n",first,second);

}

num_dati = num_dati-m+1;
```

```
farfree(Y);
fclose(fp);
 if((fp = fopen("dati.out","r")) == NULL)
   {
     sound(FREQ1);
     delay(TIME);
     nosound();
     clrscr();
     textcolor(4+128);
     gotoxy(8,10);


     cprintf("impossibile aprire il file %s","dati.out");
     *c = getchar();
     return 1;

   }


 i = 0;

 while(i < num_dati)
 {
    i++;
    fscanf(fp,"%f",&X[i]);
 }
```

```
  fclose(fp);


/*---------------------------------------------------------------------*/

  sound(FREQ0);
  delay(TIME);
  nosound();

  bordo(3,6,57,9);
  textcolor(3);
  gotoxy(2,2);
  cprintf("percentuale dati da finestrare ? ");
  cscanf("%f",&perc);
  clrscr();

  perc = perc / 100.0;

 /*------------------ finestraggio dei dati input ------------------------*/

    tot = 0.0;
    TOT = 0.0;
    num = num_dati * perc;

sound(FREQ0);
delay(TIME);
nosound();
```

```
bordo(3,6,40,16);
textcolor(1);
gotoxy(5,1);


cprintf("Menu' tipi di finestra");

textcolor(2);
gotoxy(4,3);
cprintf("1. rettangolare");

textcolor(3);
gotoxy(4,4);
cprintf("2. Bartlett");

textcolor(4);
gotoxy(4,5);
cprintf("3. Hanning");

textcolor(5);
gotoxy(4,6);
cprintf("4. Hamming");

textcolor(6);
gotoxy(4,7);
cprintf("5. Blackman");
```

```
textcolor(7);
gotoxy(4,8);
cprintf("6. Gaussiana");

textcolor(8+128);
gotoxy(4,9);
cprintf("--> ");

cscanf("%d",&scelta);

nosound();
clrscr();

/*---------------------------------------------------------------*/

   switch(scelta)
    {
 case 1 :
            i = 0;
            while(i < num_dati)
             {
               i++;
               X[i] = X[i] - media1;


               if(i <= (int)num)  W = 1.0;
```

```
            else if(i >= (int)(num_dati - num))  W = 1.0;

            X[i] *= W;

            tot += X[i];
            TOT += X[i] * X[i];

            W = 1.0;
          } break;

case 2 :
          i = 0;
          while(i < num_dati)
          {
            i++;
            X[i] = X[i] - media1;

            if(i <= (int)num)
             {
                 if(i <= (num - 1) / 2)  W = (2.0 * i) / (num - 1);
                   else  W = 2.0 - ((2.0*i)/(num-1));
             }

             else if(i >= (int)(num_dati - num))
             {
                 if((num_dati-i) <= ((num-1)/2))
                    W = (2.0*(num_dati-i))/(num-1);
```

```
            else  W = 2.0 - ((2.0*(num_dati-i))/(num-1));
          }

         X[i] *= W;
         tot += X[i];
         TOT += X[i] * X[i];

         W = 1.0;

       }  break;

case 3 :
       i = 0;
       while(i < num_dati)
        {
          i++;
          X[i] =  X[i] - media1;

          if(i <= (int)num)  W = .5 * (1.0-cos(pi * i / num));

          else if(i >= (int)(num_dati - num))
                  W = .5 * (1.0- cos(pi * (num_dati - i) / num));

          X[i] *= W;

          tot += X[i];
          TOT += X[i] * X[i];
```

```
            W = 1.0;
          } break;

case 4 :
          i = 0;
          while(i < num_dati)
          {
            i++;
            X[i] = X[i] - media1;

            if(i <= (int)num)
              W = .54 - (.46*cos((2.0*pi*i)/(num-1)));

            else if(i >= (int)(num_dati-num))
              W = .54 - (.46*cos((2.0*pi*(num_dati-i))/(num-1)));

            X[i] *= W;

            tot += X[i];
            TOT += X[i] * X[i];

            W = 1.0;
          }  break;

case 5 :
          i = 0;
```

```
            while(i < num_dati)
            {
              i++;
              if(i <= (int)num)
  W = .42 - (.5*cos((2.0*pi*i)/(num-1)))+(.08*cos((4.0*pi*i)/(num-1)));

              else if(i >= (int)(num_dati-num))
  W = .42 - (.5*cos((2.0*pi*(num_dati-i))/(num-1)))+
              (.08*cos((4.0*pi*(num_dati-i))/(num-1)));


              X[i] *= W;

              tot += X[i];
              TOT += X[i] * X[i];

              W = 1.0;
            }  break;


case 6 :
            i = 0;
            while(i < num_dati)
            {
               i++;
               if(i <= (int)num)
               W = exp(((((2.0*i)/(num-1))*((2.0*i)/(num-1)))/2.0));
```

```
                else if(i >= (int)(num_dati-num))
W = exp(((((2.0*(num_dati-i))/(num-1))*((2.0*(num_dati-i))/(num-1)))/2.0));

                X[i] *= W;

                tot += X[i];
                TOT += X[i] * X[i];

                W = 1.0;
              }  break;

  default :
              clrscr();
              sound(FREQ1);
              delay(TIME);
              nosound();
              textcolor(4+128);
              gotoxy(8,10);
              cprintf("errore nella scelta in menu'");
              *c = getchar();
              return 1;

    }

 /*----------------- media e varianza dopo finestraggio ----------*/

     media = tot / (float)num_dati;
```

```
  sound(FREQ0);
  delay(TIME);
  nosound();

  textcolor(1);
  bordo(3,6,62,13);
  gotoxy(4,1);

  cprintf("scelta tipo di varianza dei dati finestrati");

  textcolor(2);
  gotoxy(4,3);
  cprintf("1. classica con distribuzione esponenziale");

  textcolor(3);
  gotoxy(4,4);
  cprintf("2. campionaria con distribuzione di Fisher");

  textcolor(4+128);
  gotoxy(6,5);
  cprintf("--> ");

  cscanf("%d",&choice);
  clrscr();

switch(choice)
```

```
    {

      case 1 :
varianza = (TOT - ((tot * tot) / (float)num_dati)) / (float)(num_dati - 1);
           break;

      case 2 :
           varianza = (TOT / (float)(num_dati - 1));
           break;

      default :
            clrscr();
            sound(FREQ1);
            delay(TIME);
            nosound();
            textcolor(4+128);
            gotoxy(8,10);
            cprintf("errore nella scelta della varianza");
            *c = getchar();
            return 1;

    }

/*-------------------------------------------------------------------------*/

 NI = (int)(num_dati / 2.0);           /* numero frequenze indipendenti */
```

```
if((ome = (float far *) farcalloc((NI+1),sizeof(float))) == NULL)
 {
   clrscr();
   sound(FREQ1);
   delay(TIME);
   nosound();
   textcolor(4+128);
   gotoxy(8,10);
   cprintf("memoria esaurita per array delle frequenze input");
   *c = getchar();
   return 1;
 }

 /*------------------ calcolo soglia di controllo ----------------------*/

  sound(FREQ0);
  delay(TIME);
  nosound();

  textcolor(2);
  bordo(3,6,50,9);
  gotoxy(4,2);
  cprintf("percentuale di confidenza ? ");
  cscanf("%f",&po);
  clrscr();

  freq = (float)NI;
```

```
  esp = pow((1.0-po),(1.0/freq));
  zo = -((float)log(1.0 - (float)esp));
/*------------------  calcolo frequenze periodogramma ------------------*/

  i = 0;
  while(i < NI)
    {
      i++;
      ome[i] = (2.0*pi*i) / (float)num_dati;
    }


  sound(FREQ0);
  delay(TIME);
  nosound();

  bordo(3,6,66,9);
  textcolor(6);
  gotoxy(2,2);

  cprintf("Nø frequenze da esaminare (default %d) ? ",NI);
  cscanf("%d",&NI);
  clrscr();

/*-------------------- informazioni statistiche ----------------------*/

bordo(3,6,79,25);
```

```
gotoxy(3,2);
cprintf("          INFORMAZIONI STATISTICHE");
textcolor(2);

gotoxy(4,4);
cprintf("media e varianza segnale originale : %f     %f",
                                    media1,varianza1);
textcolor(3);
gotoxy(4,5);
cprintf("media e varianza dopo il finestraggio : %f   %f",
                                          media,varianza);
textcolor(4);
gotoxy(4,6);
cprintf("valore di soglia zo = %f",zo);

textcolor(5);
gotoxy(4,7);
cprintf("numero frequenze indipendenti = %d",NI);

/*------------------ calcolo periodogramma -----------------------------*/

if((out = (float far *) farcalloc((NI+1),sizeof(float))) == NULL)
  {
    clrscr();
    sound(FREQ1);
    delay(TIME);
    nosound();
```

```
  textcolor(4+128);
  gotoxy(8,10);
  cprintf("memoria esaurita per array ordinate del periodogramma");
  *c = getchar();
  return 1;
}

sound(FREQ0);
delay(TIME);
nosound();

textcolor(14);
gotoxy(4,10);
cprintf("\nnome file output ? ");
cscanf("%s",file_out);

if((fp = fopen(file_out,"w")) == NULL)
  {

    clrscr();
    sound(FREQ1);
    delay(TIME);
    nosound();
    textcolor(4+128);
    gotoxy(8,10);
    cprintf("Impossibile aprire il file %s",file_out);
    *c = getchar();
```

```
    return 1;
   }
  printf("\n\n\n");

/*------------------ calcolo ordinate ----------------------------------*/

  textcolor(12);

  i = 0;
  while(i < NI)
   {
     i++;

     sum1 = 0.0; sum2 = 0.0; sum3 = 0.0;
     sum4 = 0.0; sum5 = 0.0; sum6 = 0.0;

     j = 0;
     while(j < num_dati)
  {
    j++;
    sum1 += sin(2.0 * ome[i] * j);
    sum2 += cos(2.0 * ome[i] * j);
  }

     rap = sum1 / sum2;
     tau = atan(rap) / (2.0 * ome[i]);
```

```
    j = 0;
    while(j < num_dati)
{
  j++;

  esp1 = cos(ome[i] * (j - tau));
  esp2 = sin(ome[i] * (j - tau));

  sum3 += (X[j] * esp1);
  sum4 += esp1 * esp1;
  sum5 += (X[j] * esp2);
  sum6 += esp2 * esp2;

}

    sum3 = sum3 * sum3;
    sum5 = sum5 * sum5;
    fraz1 = sum3 / sum4;
    fraz2 = sum5 / sum6;

    pgramma = .5 * (fraz1 + fraz2);
    p_normal = pgramma / (2.0 * varianza);

    fprintf(fp,"%f\n",p_normal);
    out[i] = p_normal;

    printf("         frequenze analizzate = %d\r",i);
```

```
    }
/*-----------------------------------------------------------------------*/

fclose(fp);

/*---------------------- stampa prospetto esperimento--------------------*/

sound(FREQ0);
delay(TIME);
nosound();
textcolor(14);
gotoxy(4,17);
cprintf("\nnome file prospetto ? ");
cscanf("%s",file_prosp);

if((fp = fopen(file_prosp,"w")) == NULL)
  {
    clrscr();
    sound(FREQ1);
    delay(TIME);
    nosound();
    textcolor(4+128);
    gotoxy(8,10);
    cprintf("impossibile aprire il file %s",file_prosp);

    *c = getchar();
```

```
    return 1;
  }
fprintf(fp,"\nPROSPETTO RISULTATI ESPERIMENTO\n");

switch(scelta)
  {
    case 1 : fprintf(fp,"tipo finestra usata : Rett.\n");
       break;

    case 2 : fprintf(fp,"tipo finestra usata : Bartlett\n");
       break;

    case 3 : fprintf(fp,"tipo finestra usata : Hanning\n");
       break;

    case 4 : fprintf(fp,"tipo finestra usata : Hamming\n");
       break;

    case 5 : fprintf(fp,"tipo finestra usata : Blackman\n");
       break;

    case 6 : fprintf(fp,"tipo finestra usata : Gauss.\n");

  }

switch(choice)
  {
```

```
        case 1 : fprintf(fp,"tipo varianza usata : Classica\n");
                 break;
        case 2 : fprintf(fp,"tipo varianza usata : Campionaria\n");
                 break;

      }

    fprintf(fp,"valore soglia di controllo zo = %f\n",zo);
    fprintf(fp,"parametro di confidenza po = %f\n",po);
    fprintf(fp,"numero dati campione utilizzato = %d\n",num_dati);
    fprintf(fp,"numero frequenze indipendenti = %d\n",NI);
    fprintf(fp,"percentuale finestraggio = %f\n\n",perc);
    fprintf(fp,"FREQUENZE SIGNIFICATIVE E RELATIVI VALORI DEL
PERIODOGRAMMA :\n");

    i = 0;
    while(i < NI)
     {
       i++;
     if(out[i] >= zo)
      {
       fprintf(fp,"\nfrequenza (%d) : %f --> ordinata : %f",
                                  i,ome[i]/(2.0*pi),out[i]);
      }
     }

    fclose(fp);
```

```
 farfree(X);
 farfree(ome);
 farfree(out);

 return 0;
}

void menu()
{
  int menu1;
  char *c;

 while(1)
 {
   intestazione();

   window(3,6,80,25);

  clrscr();
  textcolor(2);
  bordo(3,2,55,13);

  gotoxy(5,1);
  textcolor(14+128);
  cprintf("MENU OPZIONI");
```

```
 gotoxy(5,3);
 textcolor(3);
 cprintf("1. lancio nuovo esperimento");

gotoxy(5,4);
textcolor(4);
cprintf("2. visualizzazione file output");

gotoxy(5,5);
textcolor(5);
cprintf("3. visualizzazione file prospetto");

gotoxy(5,6);
textcolor(6);
cprintf("4. grafico da file output");

gotoxy(5,7);
textcolor(7);
cprintf("5. Informazioni programma");

gotoxy(5,8);
textcolor(8);
cprintf("6. Quit");

gotoxy(9,9);
textcolor(9+128);
cprintf("--> ");
```

```
cscanf("%d",&menu1);
textcolor(2);

switch(menu1)
 {
   case 1 :  clrscr();
       prog();
       break;

   case 2 :  display_out();
       break;

   case 3 :  display_prosp();
       *c = getchar();
       break;

   case 4 :  display_graph();
       break;

   case 5 :  clrscr();
       sound(FREQ0);
       delay(TIME);
       nosound();
       textcolor(2);
       bordo(1,1,60,7);
       gotoxy(3,2);
```

```
      textcolor(4+128);
      cprintf("PGRAMMA.EXE  Rel. 2.0  1994");
      gotoxy(3,3);
      textcolor(4);
     cprintf("Per dettagli sull'esecuzione consultare PGRAMMA.DOC");
      gotoxy(25,4);
      cprintf("Max Brescia");
      *c = getchar();
      break;

  case 6 :  clrscr();
      sound(FREQ0);
      delay(TIME);
      nosound();
      exit(0);
      break;
  default :  clrscr();
          sound(FREQ1);
          delay(TIME);
          nosound();
          textcolor(4+128);
          gotoxy(8,10);
          cprintf("errore nella scelta!");
          *c = getchar();
          break;

}
```

```
 }
}
void display_graph()
{
     /* request auto detection */
    int gdriver = DETECT,gmode, errorcode;
    FILE *fp,*fp1;
    float offset_x,x;
    int pos_x,i,j,pos_y;
    int pp;
    float far *xx;
    char filename[30],
   ascissa[40],
   ordinata[40];
   /* int val = 0;
    char strng[5];
    char *str = "1E";
    char *key;
    int x_x = 0; */
    char *c;


  clrscr();
  textcolor(2);
  bordo(2,3,40,6);
  textcolor(3);
  gotoxy(3,2);
```

```
  cprintf("nome file ? ");
  cscanf("%s",filename);
  if((fp = fopen(filename,"r")) == NULL)
    {
      clrscr();
      sound(FREQ1);
      delay(TIME);
      nosound();
      textcolor(4+128);
      gotoxy(8,10);
      cprintf("impossibile aprire il file %s",filename);
      *c = getchar();
      exit(1);
    }

    clrscr();


i = 0;
while((fscanf(fp,"%f",&x)) != EOF)  i++;

fclose(fp);

xx = (float far *) farcalloc(i,sizeof(float));


if((fp = fopen(filename,"r")) == NULL)
```

```
  {
    fprintf(stderr,"\nimpossibile aprire il file %s",filename);
    exit(1);
  }

i = 0;
while((fscanf(fp,"%f",&xx[i])) != EOF)  i++;

fclose(fp);

printf("\nintestazione ascissa : ");
scanf("%s",ascissa);
printf("\nintestazione ordinata : ");
scanf("%s",ordinata);

  /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "..\\bgi");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk)  /* an error occurred */
    {
       printf("Graphics error: %s\n", grapherrormsg(errorcode));
       printf("Press any key to halt:");
       getch();
       exit(1); /* terminate with an error code */
    }
```

```
    setcolor(3);
    /* create a viewport */

    setviewport(0,0,640,480,CLIP_ON);


/*settextstyle(SMALL_FONT, VERT_DIR, 0);
for(x_x = 12;x_x <= 472;x_x += 30)
{
  itoa(val,key,10);
  strcat(strng,key);
  outtextxy(1,getmaxy()-x_x,strng);
  val += 3;
  strcpy(strng,str);
}   */


settextstyle(SMALL_FONT,HORIZ_DIR,0);
outtextxy(30, 10, ordinata);
outtextxy(280,getmaxy()-11,ascissa);

setcolor(5);
line(11,0,11,getmaxy() - 12);
line(11,getmaxy()-12,640,getmaxy()-12);

// DIMENSIONI UTILI PER IL GRAFICO : (12,1) -- (639,467)
```

```
offset_x = (630.00*10.0) / (float)i;
setcolor(7);
moveto(12,getmaxy()-12);
for(j=0;j<i;j++)
{
  pos_x = 12 + (int)(j * offset_x);

  pos_y = ((getmaxy() - 13) - xx[j]);


  lineto(pos_x,pos_y);

}

setcolor(4);

line(12,(int)(((getmaxy()-13)-(zo))),(getmaxx()),(int)(((getmaxy()-13)-(zo))));

getch();

scanf("%d",&pp);

closegraph();

farfree(xx);
 }
```

```
void bordo(int startx,int starty,int endx,int endy)
{
  int i;

  gotoxy(1,1);
  putch('É');

  gotoxy(2,1);
  for(i = 0;i < (endx-startx-1);i++)
     putch('Í');

  gotoxy((endx-startx),1);
  putch('»');

  gotoxy(1,(endy-starty));
  putch('È');

  gotoxy(2,(endy-starty));
  for(i = 0;i < (endx-startx-1);i++)
     putch('Í');

  gotoxy((endx-startx),(endy-starty));
  putch('¼');

  for(i = 2;i < endy-starty;i++)
    {
     gotoxy(1,i);
```

```
      putch('º');

      gotoxy(endx-startx,i);
      putch('º');
    }
}

int display_out()
{
  char riga1[10],riga2[2],riga3[10];
  FILE *p;
  char *c;
  int count;
  char file_out[20];

  clrscr();
  sound(FREQ0);
  delay(TIME);
  nosound();
  textcolor(2);
  bordo(2,3,40,6);
  textcolor(3);
  gotoxy(3,2);
  cprintf("nome file ? ");
  cscanf("%s",file_out);

  if((p = fopen(file_out,"r")) == NULL)
```

```
 {
   clrscr();
   sound(FREQ1);
   delay(TIME);
   nosound();
   textcolor(4+128);
   gotoxy(8,10);
   cprintf("impossibile aprire il file %s",file_out);
   *c = getchar();
   return 1;
 }

clrscr();

bordo(3,3,55,6);
gotoxy(4,2);
cprintf("per interrompere la visualizzazione digitare s");
*c = getchar();

window(1,1,80,25);
clrscr();
count = 0;

while((fscanf(p,"%s %s %s\n",riga1,riga2,riga3)) != EOF)
 {
   count++;
```

```
      if(count <= 24)
   {
     printf("%s %s %s\n",riga1,riga2,riga3);
   }

      else
    {
      count = 1;
      if((*c = getchar()) == 's')
        {
            return 0;
        }
    }
    }


  fclose(p);

  return 0;
}

int display_prosp()
{
  char r1[11],r2[13],r3[12],r4[10],r5[8];
  char r6[8],r7[13],r8[9];
  FILE *p;
  char *c;
```

```
int count;
char file_prosp[20];
clrscr();
sound(FREQ0);
delay(TIME);
nosound();
textcolor(2);
bordo(2,3,40,6);
textcolor(3);
gotoxy(3,2);
cprintf("nome file ? ");
cscanf("%s",file_prosp);

if((p = fopen(file_prosp,"r")) == NULL)
 {

   clrscr();
   sound(FREQ1);
   delay(TIME);
   nosound();
   textcolor(4+128);
   gotoxy(8,10);
   cprintf("impossibile aprire il file %s",file_prosp);
   *c = getchar();
   return 1;
 }
```

```
  window(1,1,80,25);
  clrscr();
  count = 11;
  textcolor(3);

/* 2 */  fscanf(p,"\n%s %s %s",r1,r2,r3);
   gotoxy(1,2);
   cprintf("%s %s %s",r1,r2,r3);
/* 3 */  fscanf(p,"\n%s %s %s %s %s",r1,r2,r3,r4,r5);
   gotoxy(1,3);
   cprintf("%s %s %s %s %s",r1,r2,r3,r4,r5);
/* 4 */     fscanf(p,"\n%s %s %s %s %s",r1,r2,r3,r4,r5);
   gotoxy(1,4);
   cprintf("%s %s %s %s %s",r1,r2,r3,r4,r5);
/* 5 */  fscanf(p,"\n%s %s %s %s %s %s %s",r1,r2,r3,r4,r5,r6,r7);
   gotoxy(1,5);
   cprintf("%s %s %s %s %s %s %s",r1,r2,r3,r4,r5,r6,r7);
/* 6 */  fscanf(p,"\n%s %s %s %s %s %s",r1,r2,r3,r4,r5,r6);
   gotoxy(1,6);
   cprintf("%s %s %s %s %s %s",r1,r2,r3,r4,r5,r6);
/* 7 */  fscanf(p,"\n%s %s %s %s %s %s",r1,r2,r3,r4,r5,r6);
   gotoxy(1,7);
   cprintf("%s %s %s %s %s %s",r1,r2,r3,r4,r5,r6);
/* 8 */  fscanf(p,"\n%s %s %s %s %s",r1,r2,r3,r4,r5);
   gotoxy(1,8);
   cprintf("%s %s %s %s %s",r1,r2,r3,r4,r5);
/* 9 */  fscanf(p,"\n%s %s %s %s",r1,r2,r3,r4);
```

```
    gotoxy(1,9);
    cprintf("%s %s %s %s",r1,r2,r3,r4);
  /* 11 */ fscanf(p,"\n\n%s %s %s %s %s %s %s %s\n\n",r1,r2,r3,r4,r5,r6,r7,r8);
    gotoxy(1,10);
    cprintf("%s %s %s %s %s %s %s %s",r1,r2,r3,r4,r5,r6,r7,r8);

   textcolor(4);

  while((fscanf(p,"%s %s %s %s %s %s %s %s\n",r1,r2,r3,r4,r5,r6,r7,r8)) !=
EOF)
   {
    count++;

    if(count <= 24)
     {
      gotoxy(1,count);
      cprintf("%s %s %s %s %s %s %s %s",r1,r2,r3,r4,r5,r6,r7,r8);
     }

    else
     {
      count = 2;
      *c = getchar();
     }
  }

   fclose(p);
```

```
 return 0;

}
```

APPENDICE E

CODICE RELATIVO ALL'IMPLEMENTAZIONE DEL MODELLO PERCETTRONE MULTISTRATO CON ALGORITMO DI BACK-PROPAGATION PER IL RICONOSCIMENTO DEI CICLI DI MILANKOVITCH

```
/*************************************************************************
*
*               MULTILAYER PERCEPTRON
*
*                 ALGORITMO DI BACK PROPAGATION
*
*        sorgente in c++ 3.0     (versione per MSDOS)
*
*
*       Uso : pms -en file.cnf file.cmd
*                                              Max brescia
*
*************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
```

```
#include <alloc.h>

#define   ESC          27
#define   ERRORLEVEL    0.02
#define   ITEMS        8
#define   MAXWEIGHT     ((float)0.3)
#define   SCALEWEIGHT   ((float)32767)

   /* tipi e prototipi per la memorizzazione dinamica degli array */

typedef float far *PFLOAT;
typedef PFLOAT VECTOR;
typedef PFLOAT *MATRIX;

void VectorAllocate(VECTOR *vector, int nCols);
void AllocateCols(PFLOAT matrix[], int nRows, int nCols);
void MatrixAllocate(MATRIX *pmatrix, int nRows, int nCols);
void MatrixFree(MATRIX matrix, int nRows);

/* def. memoria per i livelli della rete */

     /* array per input, output, pesi e target della rete */

MATRIX  out0;       /* input layer                          */
MATRIX  out1;       /* hidden1 layer                         */
MATRIX  delta1;      /* delta al livello hidden1              */
MATRIX  delw1;       /* cambio nei pesi input-hidden1         */
```

```
MATRIX  w1;          /* pesi input-hidden1                    */
MATRIX  out2;        /* livello output                        */
MATRIX  delta2;      /* delta al livello output               */
MATRIX  delw2;       /* cambio nei pesi hidden1-output        */
MATRIX  w2;          /* pesi hidden1-output                   */
MATRIX  out3;        /* 2ø livello hidden                     */
MATRIX  delta3;      /* delta secondo livello hidden          */
MATRIX  delw3;       /* cambio nei pesi hidden1-hidden2       */
MATRIX  w3;          /* pesi hidden1-hidden2                  */
MATRIX  target;      /* output target                         */
VECTOR PatternID;    /* identificatore per ogni pattern memorizzato */

int o;

float function(int f,float g)

{
  float valore;

  if (f == 1)

       valore = 1.0 / (1.0 + exp(-g));              /* sigmoide */


  if (f == 2)
      valore = tanh(g);                 /* tangente iperbolica */
```

```
   if (f == 3)

       valore = atan(g);                  /* arcotangente */

     return valore;
}

float diff_function(float x)
{
  float y;

  if (o == 1)
      y = x * (1.0 - x);

  if (o == 2)
      y = (1.0 / (cosh(x))) * (1.0 / (cosh(x)));

  if (o == 3)
      y = (1.0 / (1.0 + (x * x)));

  return(y);
}

void main(int argc, char *argv[])
{

 float eta = 0.15,              /* tasso di apprendimento per default */
```

```
    alpha = 0.075;          /* fattore momento per default       */

int nReportErrors = 100;          /* frequenza di errore di riporto */
float ErrorLevel  = ERRORLEVEL;    /* livello di errore             */
char MonitorError = 0;

float err;
float wmax,frand,valrand,seed;
float scale = SCALEWEIGHT;

float error;            /* ultimo valore errore della somma quadratica */
register int h;         /* indice livello hidden                       */
register int i;         /* indice livello input                        */
register int j;         /* indice livello output                       */
register int h3;        /* indice secondo livello hidden               */
int p,                  /* indice numero di pattern                    */
   q,                   /* indice numero di iterazione                 */
   r,                   /* indice numero di esecuzioni                 */
   nPatterns,           /* numero di patterns desiderati               */
   nInputNodes,         /* numero di nodi input                        */
   nHiddenNodes,        /* numero di nodi primo livello hidden         */
   nHiddenNodes3,       /* numero nodi secondo livello hidden          */
   nHiddenLayers,       /* numero livelli hidden                       */
   nOutputNodes,        /* numero di nodi output                       */
   nIterations,         /* numero di iterazioni desiderate             */
   label;               /* label                                       */
```

```
char tipo_features,
     tipo_target;

 FILE *fpRun,         /* file configurazione */
      *fpcmd;         /* file comandi        */

 char szResults[30];  /* vari nomi dei patterns */
 char szError[30];
 char szPattern[30];
 char szWeights[30];
 char szWeightsOut[30];

                      /* stringhe ausiliarie */
 char string0[30];

 char *progname = *argv;

 /*----------------lettura di argomenti linea di comando-----------------*/

  clrscr();

  printf("\nArtificial Neural Network\n");
  printf("  Algoritmo di Back Propagation\n");

  for (; argc > 1; argc--)
    {
     char *arg = *++argv;
```

```
    if (*arg != '-')   break;
    switch (*++arg)
    {
      case 'e': sscanf(++arg, "%d", &nReportErrors); break;
      default:  break;
    }
   }


 if (argc < 2)
   {
fprintf(stderr,"\n    Uso: %s -en nome_file.cnf nome_file.cmd\n",progname);
    fprintf(stderr,"  -en => visualizzazione errore ogni n iterazioni\n");
    fprintf(stderr,"\nPer informazioni su formattazione files esterni\n");
    fprintf(stderr,"  vedere bp.doc\n");
    exit(1);
   }
  /*------------- Apertura file.cnf per lettura----------------------*/

  if ((fpRun = fopen(*argv, "r")) == NULL)

   {
     fprintf(stderr, "%s: impossibile aprire il file %s\n",progname,*argv);
     exit(1);
   }

  /* lettura prima linea */
```

```
fscanf(fpRun,"%s %d", string0,&label);

/* ------------------inizio del ciclo di lavoro---------------------- */

  /* lettura e controllo */

  fscanf(fpRun,
"%s %d\n%s %c\n%s %d\n%s %c\n%s %d\n%s %d\n%s %d\n%s
%d\n\n\n%s %f\n%s %f\n%s %f\n%s %f\n%s %f",
  string0,
  &nInputNodes,        /* numero di nodi input */
  string0,
  &tipo_features,          /* tipo dati dei patterns */
  string0,
  &nOutputNodes,        /* numero di nodi output */
  string0,
  &tipo_target,        /* tipo dati target */
  string0,
  &nPatterns,          /* numero patterns input */
  string0,
  &nHiddenLayers,     /* numero livelli hidden */
  string0,
  &nHiddenNodes,       /* nø di neuroni primo livello hidden */
  string0,
  &nHiddenNodes3,   /* nø di neuroni secondo livello hidden */
  string0,
  &ErrorLevel,      /* tolleranza output */
```

```
    string0,
    &eta,             /* tasso di apprendimento */
    string0,
    &alpha,       /* fattore momento */
    string0,
    &seed,      /* valore random */
    string0,
    &valrand);   /* range valori random */

fclose(fpRun);

  /*------------allocazione di memoria dinamica per tutti i dati-------*/

  MatrixAllocate(&out0,       nPatterns,      nInputNodes);
  MatrixAllocate(&out1,       nPatterns,      nHiddenNodes);
  MatrixAllocate(&out2,       nPatterns,      nOutputNodes);
  MatrixAllocate(&delta2,     nPatterns,      nOutputNodes);
  MatrixAllocate(&delta1,     nPatterns,      nHiddenNodes);
  MatrixAllocate(&delw1,      nHiddenNodes,   nInputNodes + 1);
  MatrixAllocate(&w1,         nHiddenNodes,   nInputNodes + 1);

if (nHiddenNodes3 != 0)
 {

  MatrixAllocate(&out3,       nPatterns,      nHiddenNodes3);
  MatrixAllocate(&delta3,     nPatterns,      nHiddenNodes3);
  MatrixAllocate(&delw3,      nHiddenNodes3,  nHiddenNodes + 1);
```

```
 MatrixAllocate(&w3,        nHiddenNodes3,  nHiddenNodes + 1);
 MatrixAllocate(&w2,        nOutputNodes,   nHiddenNodes3 + 1);
 MatrixAllocate(&delw2,     nOutputNodes,   nHiddenNodes3 + 1);
}

else
  {
   MatrixAllocate(&w2,      nOutputNodes,   nHiddenNodes + 1);
   MatrixAllocate(&delw2,   nOutputNodes,   nHiddenNodes + 1);
  }

 MatrixAllocate(&target,    nPatterns,      nOutputNodes);
 VectorAllocate(&PatternID, nPatterns);


/*-----------------lettura file.cmd---------------------------*/

if ((fpcmd = fopen(*++argv,"r")) == NULL)
 {
   fprintf(stderr,"\nimpossibile aprire il file %s\n",*argv);
   exit(1);
 }

 fscanf(fpcmd,"%s %s\n %s %s\n %s %d\n %s %s\n %s %s\n %s %s",
 string0,
 szWeightsOut,
 string0,
```

```
 szPattern,
 string0,
 &nIterations,
 string0,
 szResults,
 string0,
 szError,
 string0,
 szWeights);

 fclose(fpcmd);


/* generazione pesi iniziali RANDOM o contenuti in file esterno */


 if ((wmax = valrand) == 0)
   wmax = MAXWEIGHT;

 if (strcmp(szWeights,"RANDOM") == 0)
   {

   /* pesi random */

     srand(seed);

     for(h=0;h < nHiddenNodes;h++)
```

```
{
  for(i=0;i < nInputNodes+1;i++)
    {
      frand = rand();

      w1[h][i] = wmax * (1.0 - 2 * frand/scale);
      delw1[h][i] = 0.0;
    }
}


    if (nHiddenNodes3 != 0)
     {


 for(h3 = 0;h3 < nHiddenNodes3;h3++)
  {
    for(h = 0;h < nHiddenNodes+1;h++)
      {
          frand = rand();

          w3[h3][h] = wmax * (1.0 - 2 * frand/scale);
          delw3[h3][h] = 0.0;
      }
  }

 for(j = 0;j < nOutputNodes;j++)
```

```
  {
    for(h3 = 0;h3 < nHiddenNodes3+1;h3++)
      {
          frand = rand();

          w2[j][h3] = wmax * (1.0 - 2 * frand/scale);
          delw2[j][h3] = 0.0;
      }
  }
   }

  else
{

 for(j = 0;j < nOutputNodes;j++)
   {
     for(h = 0;h < nHiddenNodes+1;h++)
         {
           frand = rand();

           w2[j][h] = wmax * (1.0 - 2 * frand/scale);
           delw2[j][h] = 0.0;
         }
   }
}
 }
```

```
  else
   {

    /*----------lettura delle matrici dei pesi iniziali------------------*/

    if ((fpcmd = fopen(szWeights,"r")) == NULL)
     {
fprintf(stderr, "\n\n%s: impossibile aprire %s\n\n", progname, szWeights);
       exit(1);
     }


   /* lettura dei pesi input-hidden */
   for (h = 0; h < nHiddenNodes; h++)
      for(i = 0; i <= nInputNodes; i++)
  {
    fscanf(fpcmd, "%f", &w1[h][i]);
    delw1[h][i] = 0.0;
  }

if (nHiddenNodes3 != 0)
  {

   /* lettura pesi hidden1-hidden2 */

   for (h3 = 0; h3 < nHiddenNodes3; h3++)
     for(h = 0; h <= nHiddenNodes; h++)
```

```
    {
 fscanf(fpcmd,"%f",&w3[h3][h]);
 delw3[h3][h] = 0.0;
    }


 /* lettura pesi hidden2-output */

 for (j = 0; j < nOutputNodes; j++)
    for (h3 = 0; h3 <= nHiddenNodes3; h3++)
{
 fscanf(fpcmd, "%f", &w2[j][h3]);
 delw2[j][h3] = 0.0;
}

}    /* endif */

else
  {
    /* lettura pesi hidden1-output */

    for(j = 0; j < nOutputNodes; j++)
 for(h = 0; h <= nHiddenNodes; h++)
  {
    fscanf(fpcmd,"%f",&w2[j][h]);
    delw2[j][h] = 0.0;
  }
```

```
   }

     fclose(fpcmd);

}

/*----------lettura di tutti i patterns da imparare--------------------*/

if ((fpcmd = fopen(szPattern, "r")) == NULL)
  {
fprintf(stderr,"%s: impossibile aprire il file %s\n",progname,szPattern);
   exit(1);
  }

for (p = 0; p < nPatterns; p++)
  {
    for (i = 0; i < nInputNodes; i++)
   if (fscanf(fpcmd, "%f", &out0[p][i]) != 1)
   goto ALLPATTERNSREAD;

    /* lettura degli output target per lettura patterns input */

    for (j = 0; j < nOutputNodes; j++)
    fscanf(fpcmd, "%f", &target[p][j]);

    /* lettura degli identificatori di ogni pattern */
```

```
    fscanf(fpcmd, "%f", &PatternID[p]);

  }

 ALLPATTERNSREAD:
 fclose(fpcmd);
 if (p < nPatterns)
  {
    fprintf(stderr,"%s: %d maggiori di %d patterns letti\n",
                                        progname,p,nPatterns);
    nPatterns = p;
  }


  /* apertura output file di errore */


  if ((fpcmd = fopen(szError, "w")) == NULL)
    {
      fprintf(stderr, "%s: impossibile aprire il file %s\n",
                                          progname,szError);
      exit(1);
    }

fprintf(stderr,nIterations > 1 ? "Sto provando...\n" : "Sto testando...\n");

  /*---------------- inizio loop iterazioni ----------------------------*/
```

```
printf("\nINSERISCI IL TIPO FUNZIONE DI ATTIVAZIONE:\n");
printf("\npossibili opzioni: \n1.logistica\n2.tanh\n3.arctan\n");
scanf("%d",&o);


printf("\n\nSe si desidera interrompere l'esecuzione digitare Ctrl-C");
printf("\n  Se si desidera riavviare l'apprendimento a partire\n");
printf("    dall'iterazione interrotta,inserire il nome del file\n");
printf("    temp.wei in %s come nuovo file pesi input\n\n",*argv);


for (q = 0; q < nIterations; q++)
 {
   for (p = 0; p < nPatterns; p++)
    {

/*--------------------livello hidden1-------------------------------*/

/* somma input a livello hidden su tutte le combinazioni peso-input */

for (h = 0; h < nHiddenNodes; h++)
  {
    float sum = w1[h][nInputNodes];           /*inizio col bias*/
    for (i = 0; i < nInputNodes; i++)
         sum += w1[h][i] * out0[p][i];

    out1[p][h] = function(o,sum);
```

```
   }

if (nHiddenNodes3 != 0)
  {

/*----------------------livello hidden2-----------------------------*/

for(h3 = 0; h3 < nHiddenNodes3; h3++)
  {
    float sum = w3[h3][nHiddenNodes];
    for(h = 0; h < nHiddenNodes; h++)
   sum += w3[h3][h] *out1[p][h];

    out3[p][h3] = function(o,sum);
  }

/*-------------------livello output----------------------------*/

  for (j = 0; j < nOutputNodes; j++)
    {
       float sum = w2[j][nHiddenNodes3];
       for (h3 = 0; h3 < nHiddenNodes3; h3++)
                    sum += w2[j][h3] * out3[p][h3];

       out2[p][j] = function(o,sum);
    }
```

```
}    /* endif */

else
  {

  /*-----------------livello output--------------------------------*/

    for(j = 0; j < nOutputNodes; j++)
{
  float sum = w2[j][nHiddenNodes];
  for(h = 0; h < nHiddenNodes; h++)
         sum += w2[j][h] * out1[p][h];

  out2[p][j] = function(o,sum);
}
 }

/*-------------------delta output---------------------------------*/

   /* calcolo dei delta per ogni unita' output per un dato pattern */

 for (j = 0; j < nOutputNodes; j++)

  delta2[p][j] = (target[p][j]-out2[p][j]) * diff_function(out2[p][j]);
if (nHiddenNodes3 != 0)
{
```

```
/*-------------------delta hidden2--------------------------------*/

 for(h3 = 0; h3 < nHiddenNodes3; h3++)
   {
     float sum = 0.0;
     for(j = 0; j < nOutputNodes; j++)
           sum += delta2[p][j] * w2[j][h3];

     delta3[p][h3] = sum * diff_function(out3[p][h3]);
   }

/*---------------delta hidden1--------------------------------*/

for(h = 0; h < nHiddenNodes; h++)
 {
   float sum = 0.0;
   for(h3 = 0; h3 < nHiddenNodes3; h3++)
    sum += delta3[p][h3] * w3[h3][h];

   delta1[p][h] = sum * diff_function(out1[p][h]);
 }

}   /* endif */

 else
  {
```

```
   /*-------------------delta hidden1--------------------------*/

  for (h = 0; h < nHiddenNodes; h++)
   {
     float sum = 0.0;
     for (j = 0; j < nOutputNodes; j++)
           sum += delta2[p][j] * w2[j][h];

     delta1[p][h] = sum * diff_function(out1[p][h]);
   }
   }

}

if (nHiddenNodes3 != 0)
 {

/*----------------adattamento pesi hidden2-output--------------------*/

for(j = 0; j < nOutputNodes; j++)
  {
    float dw;                   /* peso delta */
    float sum = 0.0;

 /* somma dei delta per ogni nodo output per un'epoca */

 for(p = 0; p < nPatterns; p++)
```

```
    sum += delta2[p][j];

  /* calcolo nuovo peso bias per ogni unita' output */

  dw = eta * sum + alpha * delw2[j][nHiddenNodes3];
  w2[j][nHiddenNodes3] += dw;
  delw2[j][nHiddenNodes3] = dw;

  /* calcolo nuovi pesi */

  for(h3 = 0; h3 < nHiddenNodes3; h3++)
    {
      float sum = 0.0;
      for(p = 0; p < nPatterns; p++)
   sum += delta2[p][j] * out3[p][h3];

      dw = eta * sum + alpha * delw2[j][h3];
      w2[j][h3] += dw;
      delw2[j][h3] = dw;
    }
 }


  /* adattamento pesi hidden2-hidden1 */

  for(h3 = 0; h3 < nHiddenNodes3; h3++)
    {
```

```
      float dw;
      float sum = 0.0;


/* somma dei delta per ogni nodo del secondo livello hidden per epoca */

      for(p = 0; p < nPatterns; p++)
    sum += delta3[p][h3];


/* calcolo nuovo peso bias per ogni unita' del secondo livello hidden */

      dw = eta * sum + alpha * delw3[h3][nHiddenNodes];
      w3[h3][nHiddenNodes] += dw;
      delw3[h3][nHiddenNodes] = dw;
        /* calcolo nuovi pesi */

      for(h = 0; h < nHiddenNodes; h++)
   {
    float sum = 0.0;
    for(p = 0; p < nPatterns; p++)
        sum += delta3[p][h3] * out1[p][h];

    dw = eta * sum + alpha * delw3[h3][h];
    w3[h3][h] += dw;
    delw3[h3][h] = dw;
   }
```

```
  }
}     /* endif */

 else
  {

   /*-------------adattamento dei pesi hidden1-output-------------*/

for (j = 0; j < nOutputNodes; j++)
 {
  float dw;                              /* peso delta */
  float sum = 0.0;

  /* somma dei delta per ogni nodo output per un'epoca */

  for (p = 0; p < nPatterns; p++)
sum += delta2[p][j];

  /* calcolo nuovo peso bias per ogni unita' output */

  dw = eta * sum + alpha * delw2[j][nHiddenNodes];
  w2[j][nHiddenNodes] += dw;
  delw2[j][nHiddenNodes] = dw;               /* delta per il bias */

  /* calcolo nuovi pesi */

  for (h = 0; h < nHiddenNodes; h++)
```

```
    {
  float sum = 0.0;
  for (p = 0; p < nPatterns; p++)
       sum += delta2[p][j] * out1[p][h];

  dw = eta * sum + alpha * delw2[j][h];
  w2[j][h] += dw;
  delw2[j][h] = dw;
    }
  }
}      /* end else */

  /*-----------adattamento pesi input-hidden1-------------------------*/

  for (h = 0; h < nHiddenNodes; h++)
    {
      float dw;
      float sum = 0.0;
      for (p=0; p < nPatterns; p++)
   sum += delta1[p][h];


      /* calcolo nuovo peso bias per ogni unita' hidden */

      dw = eta * sum + alpha * delw1[h][nInputNodes];
      w1[h][nInputNodes] += dw;
      delw1[h][nInputNodes] = dw;
```

```
    /* calcolo nuovi pesi */

    for (i = 0; i < nInputNodes; i++)
 {
  float sum = 0.0;
  for (p = 0; p < nPatterns; p++)
        sum += delta1[p][h] * out0[p][i];

  dw = eta * sum + alpha * delw1[h][i];
  w1[h][i] += dw;
  delw1[h][i] = dw;
 }
  }
/*----------------controllo tastiera---------------------------------*/

if (kbhit())
 {
   int c = getchar();
   if ((c = toupper(c)) == 'E')
  MonitorError++;
  else if (c == ESC)
      break;
 }

 /*-----------------errore di somma quadratica-------------------------*/

fprintf(fpRun,"\nVALORI ERRORE SU TUTTI I PATTERNS\n\n\n");
```

```
 if (MonitorError || (q % nReportErrors == 0))
   {
     for (p = 0, error = 0.0; p < nPatterns; p++)
 {
   for (j = 0;j < nOutputNodes; j++)
     {
       float temp = target[p][j] - out2[p][j];
       error += temp * temp;
     }
 }


  /* errore medio per nodo su tutti i patterns */

  error /= (nPatterns * nOutputNodes);


  /* stampa il numero d'iterazione ed il valore errore */

fprintf(stderr,"Iterazione %5d/%-5d   Errore %f\r",q,nIterations,error);

  MonitorError = 0;


  if (q % nReportErrors == 0)
     fprintf(fpcmd,"\niterazione %d    errore : %f\n\n",q,error);
```

```
/*------------------stampa pesi finali---------------------------------*/

if ((fpRun = fopen(szWeightsOut,"w")) == NULL)

   {
fprintf(stderr, "%s: impossibile scrivere il file %s\n",
              prognname,szWeightsOut);
     exit(1);
   }

fprintf(fpRun,"\n  STATO DELLA RETE ALL'ITERAZIONE %d\n",q);
fprintf(fpRun,"\n\n  sono compresi i neuroni bias!\n\n\n");

fprintf(fpRun,"  PESI INPUT-HIDDEN1 :\n\n");

for (h = 0; h < nHiddenNodes; h++)
   for (i = 0; i <= nInputNodes; i++)
fprintf(fpRun,"   in_hid1[%d][%d] = %g\n\n",i,h,w1[h][i]);


if (nHiddenNodes3 != 0)
{

  fprintf(fpRun,"  PESI HIDDEN1-HIDDEN2 :\n\n");

  for(h3 = 0; h3 < nHiddenNodes3; h3++)
   for(h = 0; h < nHiddenNodes+1; h++)
```

```
 fprintf(fpRun,"   hid1_hid2[%d][%d] = %g\n\n",h,h3,w3[h3][h]);

  fprintf(fpRun,"  PESI HIDDEN2-OUTPUT :\n\n");

  for (j = 0;j < nOutputNodes;j++)
    for (h3 = 0;h3 < nHiddenNodes3+1;h3++)
 fprintf(fpRun,"   hid2_out[%d][%d] = %g\n\n",h3,j,w2[j][h3]);


}

 else
 {
  fprintf(fpRun,"  PESI HIDDEN1-OUTPUT :\n\n");

   for (j = 0; j < nOutputNodes; j++)
    for (h = 0; h <= nHiddenNodes; h++)
 fprintf(fpRun,"   hid1_out[%d][%d] = %g\n\n",h,j,w2[j][h]);

 }
fclose(fpRun);


if ((fpRun = fopen("temp.wei","w")) == NULL)
 {
   fprintf(stderr,"\nimpossibile aprire il file\n");
   exit(1);
```

```
}

  for(h = 0;h < nHiddenNodes;h++)
    for(i = 0;i < nInputNodes+1;i++)
fprintf(fpRun,"%9.6f\n",w1[h][i]);

 if (nHiddenNodes3 != 0)
  {
    for(h3 = 0;h3 < nHiddenNodes3;h3++)
for(h = 0;h < nHiddenNodes+1;h++)
  fprintf(fpRun,"%9.6f\n",w3[h3][h]);

    for(j = 0;j < nOutputNodes;j++)
for(h3 = 0;h3 < nHiddenNodes3+1;h3++)
  fprintf(fpRun,"%9.6f\n",w2[j][h3]);
  }

 else
   {
for(j = 0;j < nOutputNodes;j++)
  for(h = 0;h < nHiddenNodes+1;h++)
    fprintf(fpRun,"%9.6f\n",w2[j][h]);
   }


fclose(fpRun);
```

```
/*------------------stampa valori attivazione finale----------------------*/

if ((fpRun = fopen(szResults,"w")) == NULL)
 {
   fprintf(stderr, "%s: impossibile scrivere il file %s\n",
                                        progname,szResults);
   fpRun = stderr;
 }


 /* stampa vettore output finale */

 fprintf(fpRun,"\n   OUTPUT DELLA RETE ALL'ITERAZIONE %d\n\n\n",q);
 fprintf(fpRun,"parametri usati :\n");
 fprintf(fpRun,"\n     eta   = %f\n",eta);
 fprintf(fpRun,"       alpha = %f\n\n\n",alpha);


 for (p = 0; p < nPatterns; p++)
  {
    for (j = 0; j < nOutputNodes; j++)
     {
  err = target[p][j] - out2[p][j];
  if (err < 0)  err = err * -1;

  fprintf(fpRun, "  out[%d] = %f",j,out2[p][j]);
     }
```

```
    fprintf(fpRun,"  pattern %-6.0f  errore = %f\n",PatternID[p],err);
   }

 fclose(fpRun);

 if (error < ErrorLevel)
    break;
}

 }       /* end ciclo for delle iterazioni */

/*---------------fine del loop d'iterazione-----------------------------*/

for (p = 0, error = 0.0; p < nPatterns; p++)
  {
    for (j = 0; j < nOutputNodes; j++)
      {
  float temp = target[p][j] - out2[p][j];
  error += temp * temp;
      }
  }

  /* errore medio su tutti i patterns */

  error /= (nPatterns * nOutputNodes);
  /* stampa numero d'iterazione finale e valore di errore */
```

```
 fprintf(stderr,"Iterazione %5d/%-5d   Errore %f\n",q,nIterations,error);
 fprintf(fpcmd, "\niterazione finale %d       errore : %f\n\n",q,error);

   fclose(fpcmd);

/*--------------memoria dinamica libera per dati-------------------------*/

MatrixFree(out0,          nPatterns);
MatrixFree(out1,          nPatterns);
MatrixFree(delta1,        nPatterns);
MatrixFree(delw1,         nHiddenNodes);
MatrixFree(w1,            nHiddenNodes);
MatrixFree(out2,          nPatterns);
MatrixFree(delta2,        nPatterns);
MatrixFree(delw2,         nOutputNodes);
MatrixFree(w2,            nOutputNodes);

if (nHiddenNodes3 != 0)
{

 MatrixFree(out3,          nPatterns);
 MatrixFree(delta3,        nPatterns);
 MatrixFree(delw3,         nHiddenNodes3);
 MatrixFree(w3,            nHiddenNodes3);

}
```

```
MatrixFree(target,         nPatterns);
free(PatternID);

printf ("\n PER I RISULTATI CONSULTARE I SEGUENTI FILES :\n");
printf("  %s  per valori output\n",szResults);
printf("  %s  per pesi finali\n",szWeightsOut);
printf("  %s  per errori\n",szError);

}

/*-------------routines di allocazione di memoria per array--------------*/


/*spazio allocato per vettore di celle float per un vector[cols] dinamico*/


void VectorAllocate(VECTOR *vector, int nCols)
{
  if ((*vector = (VECTOR) farcalloc(nCols, sizeof(float))) == NULL)
    {
      fprintf(stderr, "VectorAllocate memoria insufficiente per i nodi\n");
      exit(1);
    }
}

/* spazio allocato per colonne (celle float) per una matrice dinamica */
```

```
void AllocateCols(PFLOAT matrix[], int nRows, int nCols)
{
  int i;
  for (i = 0; i < nRows; i++)
     VectorAllocate(&matrix[i], nCols);
}

/* spazio allocato per una matrice dinamica */

void MatrixAllocate(MATRIX *pmatrix, int nRows, int nCols)
{
  if ((*pmatrix = (MATRIX) farcalloc(nRows, sizeof(PFLOAT))) == NULL)
    {
       fprintf(stderr,"MatrixAllocate memoria insufficiente per i nodi\n");
       exit(1);
    }

  AllocateCols(*pmatrix,nRows,nCols);

 }

/* spazio libero per un array bidimensionale dinamico */

void MatrixFree(MATRIX matrix,int nRows)
  {
    int i;
    for (i = 0; i < nRows; i++)
```

```
    farfree(matrix[i]);

  farfree(matrix);
}
```

APPENDICE F

CODICE RELATIVO AL PROGRAMMA DI GENERAZIONE DEI SEGNALI SINTETICI CONTENENTI LE FREQUENZE DI MILANKOVITCH.

```
/*************************************************************************
*
*         FILE X GENERAZIONE SEGNALI DI MILANKOVITCH
*
*           RELEASE 30-05-94
*
*                                    MAX BRESCIA
*
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <math.h>
#include <ctype.h>
#include <alloc.h>

#define   MIL1   ((double)410000)
#define   MIL2   ((double)100000)
#define   MIL3   ((double)49367)
#define   MIL4   ((double)38202)
```

```
#define   MIL5   ((double)22056)
#define   MIL6   ((double)18346)
#define   SCALE  ((float)32767)
#define   SEED   ((float)454)
#define   EPS    ((float)0.45)

void sign1(long Np,int period1);
void sign2(long Np,int period1,int period2);
void sign3(long Np,int period1,int period2,int period3);
void sign4(long Np,int period1,int period2,int period3,int period4);
void sign5(long Np,int period1,int period2,int period3,int period4,int period5);
void sign6(long Np);


double pi = 3.141592654;

void main()
{
  long N;             //  NUMERO PUNTI DELLE SEQUENZE GENERATE
  int  Nf,            //  NUMERO FREQUENZE DA COMPORRE
       choice1,       //  INDICE PERIODO SCELTO
       choice2,       //          "
       choice3,       //          "
       choice4,       //          "
       choice5,       //          "
       choice6;       //          "
 clrscr();
```

```
        fprintf(stderr,"\n                              GENERAZIONE    SEGNALI    DI
MILANKOVITCH\n\n\n");
        fprintf(stderr,"numero di punti ? ");
        scanf("%ld",&N);
        fprintf(stderr,"\nnumero di frequenze da inserire  (MAX 6) ? ");
        scanf("%d",&Nf);

        if(Nf == 6)  sign6(N);

       else
       {
        fprintf(stderr,"\n\n   MENU SCELTA PERIODI :\n");
        fprintf(stderr,"1. 410000         4. 38202\n");
        fprintf(stderr,"2. 100000         5. 22056\n");
        fprintf(stderr,"3. 49367          6. 18346\n");
        fprintf(stderr,"\nscegli i periodi :   ");

        switch(Nf)
         {
           case 1 :
             fscanf(stdin,"%d",&choice1);
             sign1(N,choice1);
             break;

           case 2 :
             fscanf(stdin,"%d %d",&choice1,&choice2);
```

```
    sign2(N,choice1,choice2);
    break;

  case 3 :
    fscanf(stdin,"%d %d %d",&choice1,&choice2,&choice3);
    sign3(N,choice1,choice2,choice3);
    break;

  case 4 :
    fscanf(stdin,"%d %d %d %d",
              &choice1,&choice2,&choice3,&choice4);
    sign4(N,choice1,choice2,choice3,choice4);
    break;

  case 5 :
    fscanf(stdin,"%d %d %d %d %d",
             &choice1,&choice2,&choice3,&choice4,&choice5);
    sign5(N,choice1,choice2,choice3,choice4,choice5);
    break;

  default :
     fprintf(stderr,"\nErrore nella scelta!\n");
     fprintf(stderr,"inserire i numeri sulla stessa riga\n");
     fprintf(stderr,"separati da uno spazio!\n");
     exit(1);

}
```

```
 }
 printf("\nI segnali sono stati generati\n\nbye bye!\n");

}

                              // SIGN1

void sign1(long Np,int period1)
{
  double mk1,freq1;
  int t,punti;
  FILE *fout;
  char f[30];
  float  seed = SEED,
   frand,
   valrand,
   epsilon = EPS,
   scale = SCALE,
   T,
   tt;
  float far *v;
  float  far *quad;

  switch(period1)
   {
      case 1 :
        mk1 = MIL1;
```

```
      break;
    case 2 :
      mk1 = MIL2;
      break;
    case 3 :
      mk1 = MIL3;
      break;
    case 4 :
      mk1 = MIL4;
      break;
    case 5 :
      mk1 = MIL5;
      break;
    case 6 :
      mk1 = MIL6;
      break;
    default :
       fprintf(stderr,"\n Errore nella scelta in sign1\n");
       exit(1);

  }


freq1 = 1.0 / mk1;


/*----------------GENERAZIONE SINUSOIDI-------------------------------*/
```

```
fprintf(stderr,"\nIntervallo di campionamento : ");
scanf("%f",&T);
printf("\n");

punti = (int)(Np / T);

 v = (float far *) farcalloc(punti,sizeof(float));
 quad = (float far *) farcalloc(punti,sizeof(float));

srand(seed);

t = 0;
for(tt = 0.0;tt < Np;tt += T)
 {
   frand = rand() / scale;
   valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));
if(frand > epsilon)            // ADD NOISE
   {
     v[t] = 1.0 + sin(2.0*pi*freq1*tt) + valrand;
   }

  else
     v[t] = 1.0 + sin(2.0*pi*freq1*tt);

  t += 1;
 }
```

```
/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
  if(v[t] < 0.67)
    quad[t] = 0.0;
  if((v[t] <= 1.1) && (v[t] >= 0.67))
    quad[t] = 0.2;
  if((v[t] <= 1.53) && (v[t] > 1.1))
    quad[t] = 0.4;
  if((v[t] <= 1.96) && (v[t] > 1.53))
    quad[t] = 0.6;
  if(v[t] > 1.96)
    quad[t] = 0.8;
}

printf("\nnome file out ? ");
scanf("%s",f);
if((fout = fopen(f,"w")) == NULL)
 {
  fprintf(stderr,"\nimpossibile aprire il file %s\n",f);
  exit(1);
 }

for(t=0;t < punti;t++)
  fprintf(fout,"%f\n",quad[t]);
```

```
fclose(fout);

farfree(v);
farfree(quad);
}

                                        // SIGN2

void sign2(long Np,int period1,int period2)
{
  double mk1,mk2,freq1,freq2;
  int t,punti;
  FILE *fout;
  char f[30];
  float  seed = SEED,
   frand,
   valrand,
   epsilon = EPS,
   scale = SCALE,
   T,
   tt;
  float far *v1,*v2;


  switch(period1)
   {
```

```
   case 1 :
     mk1 = MIL1;
     break;
   case 2 :
     mk1 = MIL2;
     break;
   case 3 :
     mk1 = MIL3;
     break;
   case 4 :
     mk1 = MIL4;
     break;
   case 5 :
     mk1 = MIL5;
     break;
   case 6 :
     mk1 = MIL6;
     break;
   default :
      fprintf(stderr,"\n Errore nella scelta in sign2\n");
      exit(1);

}

switch(period2)
{
   case 1 :
```

```
      mk2 = MIL1;
      break;
    case 2 :
      mk2 = MIL2;
      break;
    case 3 :
      mk2 = MIL3;
      break;
    case 4 :
      mk2 = MIL4;
      break;
    case 5 :
      mk2 = MIL5;
      break;
    case 6 :
      mk2 = MIL6;
      break;
    default :
       fprintf(stderr,"\n Errore nella scelta in sign2\n");
       exit(1);

  }

 freq1 = 1.0 / mk1;
 freq2 = 1.0 / mk2;

fprintf(stderr,"\nIntervallo di campionamento : ");
```

```
scanf("%f",&T);
printf("\n");

punti = (int)(Np / T);

 v1 = (float far *) farcalloc(punti,sizeof(float));
 v2 = (float far *) farcalloc(punti,sizeof(float));

srand(seed);

/*--------------------GENERAZIONE PRIMA SINUSOIDE-----------------------*/

t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
   frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)           // ADD NOISE
   {
     v1[t] = 1.0 + sin(2.0*pi*freq1*tt) + valrand;
   }

  else
     v1[t] = 1.0 + sin(2.0*pi*freq1*tt);

  t += 1;
```

```
}

/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}


/*--------------------GENERAZIONE SECONDA SINUSOIDE----------------------*/

t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
   frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));
if(frand > epsilon)           // ADD NOISE
```

```
   {
     v2[t] = 1.0 + sin(2.0*pi*freq2*tt) + valrand;
   }

  else
     v2[t] = 1.0 + sin(2.0*pi*freq2*tt);

  t += 1;
}

/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
  if(v2[t] < 0.67)
    v2[t] = 0.0;
  if((v2[t] <= 1.1) && (v2[t] >= 0.67))
    v2[t] = 0.2;
  if((v2[t] <= 1.53) && (v2[t] > 1.1))
    v2[t] = 0.4;
  if((v2[t] <= 1.96) && (v2[t] > 1.53))
    v2[t] = 0.6;
  if(v2[t] > 1.96)
    v2[t] = 0.8;
}

/*----------------------GENERAZIONE SEGNALE FINALE--------------------*/
```

```
for(t=0;t < punti;t++)
  v2[t] += v1[t];              // SOMMA DELLE ONDE QUADRE

for(t=0;t < punti;t++)         // RINORMALIZZAZIONE
{
  if(v2[t] <= 0.2)
    v2[t] = 0.0;
  if((v2[t] <= 0.6) && (v2[t] > 0.2))
    v2[t] = 0.2;
  if((v2[t] <= 1.0) && (v2[t] > 0.6))
    v2[t] = 0.4;
  if((v2[t] <= 1.4) && (v2[t] > 1.0))
    v2[t] = 0.6;
  if(v2[t] > 1.4)
    v2[t] = 0.8;
}

printf("\nnome file out ? ");
scanf("%s",f);
if((fout = fopen(f,"w")) == NULL)
 {
  fprintf(stderr,"\nimpossibile aprire il file %s\n",f);
  exit(1);
 }

for(t=0;t < punti;t++)
```

```
  fprintf(fout,"%f\n",v2[t]);

fclose(fout);

farfree(v1);
farfree(v2);
}
                                        // SIGN3


void sign3(long Np,int period1,int period2,int period3)
{
  double mk1,mk2,mk3,freq1,freq2,freq3;
  int t,punti;
  FILE *fout;
  char f[30];
  float  seed = SEED,
   frand,
   valrand,
   epsilon = EPS,
   scale = SCALE,
   T,
   tt;
  float  far *v1;
  float  far *v2;

  switch(period1)
```

```
{
   case 1 :
     mk1 = MIL1;
     break;
   case 2 :
     mk1 = MIL2;
     break;
   case 3 :
     mk1 = MIL3;
     break;
   case 4 :
     mk1 = MIL4;
     break;
   case 5 :
     mk1 = MIL5;
     break;
   case 6 :
     mk1 = MIL6;
     break;
   default :
      fprintf(stderr,"\n Errore nella scelta in sign3\n");
      exit(1);

}


switch(period2)
```

```
{
   case 1 :
     mk2 = MIL1;
     break;
   case 2 :
     mk2 = MIL2;
     break;
   case 3 :
     mk2 = MIL3;
     break;
   case 4 :
     mk2 = MIL4;
     break;
   case 5 :
     mk2 = MIL5;
     break;
   case 6 :
     mk2 = MIL6;
     break;
   default :
      fprintf(stderr,"\n Errore nella scelta in sign3\n");
      exit(1);

}

  switch(period3)
{
```

```
    case 1 :
      mk3 = MIL1;
      break;

    case 2 :
      mk3 = MIL2;
      break;
    case 3 :
      mk3 = MIL3;
      break;
    case 4 :
      mk3 = MIL4;
      break;
    case 5 :
      mk3 = MIL5;
      break;
    case 6 :
      mk3 = MIL6;
      break;
    default :
       fprintf(stderr,"\n Errore nella scelta in sign3\n");
       exit(1);

  }


freq1 = 1.0 / mk1;
```

```
 freq2 = 1.0 / mk2;
 freq3 = 1.0 / mk3;


fprintf(stderr,"\nIntervallo di campionamento : ");
scanf("%f",&T);
printf("\n");

punti = (int)(Np / T);

 v1 = (float far *) farcalloc(punti,sizeof(float));
 v2 = (float far *) farcalloc(punti,sizeof(float));


srand(seed);

/*-----------------GENERAZIONE PRIMA E SECONDA SINUSOIDE----------------
--*/

t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
  frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)           // ADD NOISE
{
```

```
 v1[t] = 1.0 + sin(2.0*pi*freq1*tt) + valrand;
}

 else
      v1[t] = 1.0 + sin(2.0*pi*freq1*tt);

  frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)           // ADD NOISE
{
 v2[t] = 1.0 + sin(2.0*pi*freq2*tt) + valrand;
}

  else
       v2[t] = 1.0 + sin(2.0*pi*freq2*tt);

  t += 1;
}

/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
```

```
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}

for(t=0;t < punti;t++)
{
  if(v2[t] < 0.67)
    v2[t] = 0.0;
  if((v2[t] <= 1.1) && (v2[t] >= 0.67))
    v2[t] = 0.2;
  if((v2[t] <= 1.53) && (v2[t] > 1.1))
    v2[t] = 0.4;

  if((v2[t] <= 1.96) && (v2[t] > 1.53))
    v2[t] = 0.6;
  if(v2[t] > 1.96)
    v2[t] = 0.8;
}

for(t=0;t < punti;t++)
  v2[t] += v1[t];
/*-------------------------GENERAZIONE TERZA SINUSOIDE-----------------*/
```

```
t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
  frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)            // ADD NOISE
{
 v1[t] = 1.0 + sin(2.0*pi*freq3*tt) + valrand;
}

  else
       v1[t] = 1.0 + sin(2.0*pi*freq3*tt);

  t += 1;
}

/*----------------------QUADRATURA-------------------------------------*/

for(t=0;t < punti;t++)
{
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
```

```
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}

for(t=0;t < punti;t++)            //  SOMMA FINALE
 v2[t] += v1[t];

for(t=0;t < punti;t++)            //  NORMALIZZAZIONE FINALE
{
  if(v2[t] <= 0.2)
    v2[t] = 0.0;
  if((v2[t] <= 0.8) && (v2[t] > 0.2))
    v2[t] = 0.2;
  if((v2[t] <= 1.4) && (v2[t] > 0.8))
    v2[t] = 0.4;
  if((v2[t] <= 2.0) && (v2[t] > 1.4))
    v2[t] = 0.6;
  if(v2[t] > 2.0)
    v2[t] = 0.8;
}


printf("\nnome file out ? ");
scanf("%s",f);
```

```
if((fout = fopen(f,"w")) == NULL)
 {
  fprintf(stderr,"\nimpossibile aprire il file %s\n",f);
  exit(1);
 }

for(t=0;t < punti;t++)
  fprintf(fout,"%f\n",v2[t]);

fclose(fout);

farfree(v1);
farfree(v2);
}
                        // SIGN4


void sign4(long Np,int period1,int period2,int period3,int period4)
{
  double mk1,mk2,mk3,mk4,freq1,freq2,freq3,freq4;
  int t,punti;
  FILE *fout;
  char f[30];
  float  seed = SEED,
   frand,
   valrand,
   epsilon = EPS,
```

```
 scale = SCALE,
 T,
 tt;
float  far *v1;
float  far *v2;


switch(period1)
 {
    case 1 :
      mk1 = MIL1;
      break;
    case 2 :
      mk1 = MIL2;
      break;
    case 3 :
      mk1 = MIL3;
      break;
    case 4 :
      mk1 = MIL4;
      break;
    case 5 :
      mk1 = MIL5;
      break;
    case 6 :
      mk1 = MIL6;
      break;
```

```
   default :
      fprintf(stderr,"\n Errore nella scelta in sign4\n");
      exit(1);

}


switch(period2)
{
   case 1 :
     mk2 = MIL1;
     break;
   case 2 :
     mk2 = MIL2;
     break;
   case 3 :
     mk2 = MIL3;
     break;
   case 4 :
     mk2 = MIL4;
     break;
   case 5 :
     mk2 = MIL5;
     break;
   case 6 :
     mk2 = MIL6;
     break;
```

```
   default :
      fprintf(stderr,"\n Errore nella scelta in sign4\n");
      exit(1);

}

  switch(period3)
{
   case 1 :
     mk3 = MIL1;
     break;
   case 2 :
     mk3 = MIL2;
     break;
   case 3 :
     mk3 = MIL3;
     break;
   case 4 :
     mk3 = MIL4;
     break;
   case 5 :
     mk3 = MIL5;
     break;
   case 6 :
     mk3 = MIL6;
     break;
   default :
```

```
      fprintf(stderr,"\n Errore nella scelta in sign4\n");
      exit(1);

}


switch(period4)
{
   case 1 :
     mk4 = MIL1;
     break;
   case 2 :
     mk4 = MIL2;
     break;
   case 3 :
     mk4 = MIL3;
     break;
   case 4 :
     mk4 = MIL4;
     break;
   case 5 :
     mk4 = MIL5;
     break;
   case 6 :
     mk4 = MIL6;
     break;
   default :
```

```
        fprintf(stderr,"\n Errore nella scelta in sign4\n");
        exit(1);

   }


 freq1 = 1.0 / mk1;
 freq2 = 1.0 / mk2;
 freq3 = 1.0 / mk3;
 freq4 = 1.0 / mk4;


fprintf(stderr,"\nIntervallo di campionamento : ");
scanf("%f",&T);
printf("\n");

punti = (int)(Np / T);

v1 = (float far *) farcalloc(punti,sizeof(float));
v2 = (float far *) farcalloc(punti,sizeof(float));


srand(seed);

/*----------------GENERAZIONE PRIMA E SECONDA SINUSOIDE-------------*/

t = 0;
```

```
for(tt=0.0;tt < Np;tt+=T)
{
  frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)            // ADD NOISE
{
 v1[t] = 1.0 + sin(2.0*pi*freq1*tt) + valrand;
}
  else
       v1[t] = 1.0 + sin(2.0*pi*freq1*tt);

   frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)            // ADD NOISE
{
 v2[t] = 1.0 + sin(2.0*pi*freq2*tt) + valrand;
}

  else
       v2[t] = 1.0 + sin(2.0*pi*freq2*tt);

  t += 1;
}

/*----------------------QUADRATURA-----------------------------------*/
```

```
for(t=0;t < punti;t++)
{
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}

for(t=0;t < punti;t++)
{
  if(v2[t] < 0.67)
    v2[t] = 0.0;
  if((v2[t] <= 1.1) && (v2[t] >= 0.67))
    v2[t] = 0.2;
  if((v2[t] <= 1.53) && (v2[t] > 1.1))
    v2[t] = 0.4;
  if((v2[t] <= 1.96) && (v2[t] > 1.53))
    v2[t] = 0.6;
  if(v2[t] > 1.96)
    v2[t] = 0.8;
```

```
}

for(t=0;t < punti;t++)
  v2[t] += v1[t];

/*----------------------GENERAZIONE TERZA SINUSOIDE------------------*/

t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
  frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)            // ADD NOISE
{
 v1[t] = 1.0 + sin(2.0*pi*freq3*tt) + valrand;
}

  else
       v1[t] = 1.0 + sin(2.0*pi*freq3*tt);

  t += 1;
}

/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
```

```
{
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}


for(t=0;t < punti;t++)
  v2[t] += v1[t];

/*------------------------GENERAZIONE QUARTA SINUSOIDE-------------------*/

t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
  frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)           // ADD NOISE
{
```

```
 v1[t] = 1.0 + sin(2.0*pi*freq4*tt) + valrand;
}

  else
       v1[t] = 1.0 + sin(2.0*pi*freq4*tt);

  t += 1;
}

/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
   if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}


for(t=0;t < punti;t++)              //  SOMMA FINALE
```

```
  v2[t] += v1[t];

for(t=0;t < punti;t++)              //  NORMALIZZAZIONE FINALE
{
  if(v2[t] <= 0.4)
    v2[t] = 0.0;
  if((v2[t] <= 1.2) && (v2[t] > 0.4))
    v2[t] = 0.2;
  if((v2[t] <= 2.0) && (v2[t] > 1.2))
    v2[t] = 0.4;
  if((v2[t] <= 2.8) && (v2[t] > 2.0))
    v2[t] = 0.6;
  if(v2[t] > 2.8)
    v2[t] = 0.8;
}


printf("\nnome file out ? ");
scanf("%s",f);
if((fout = fopen(f,"w")) == NULL)
 {
  fprintf(stderr,"\nimpossibile aprire il file %s\n",f);
  exit(1);
 }

for(t=0;t < punti;t++)
  fprintf(fout,"%f\n",v2[t]);
```

```
fclose(fout);

farfree(v1);
farfree(v2);
}
                        // SIGN5


void sign5(long Np,int period1,int period2,int period3,int period4,int period5)
{
  double mk1,mk2,mk3,mk4,mk5,freq1,freq2,freq3,freq4,freq5;
  int t,punti;
  FILE *fout;
  char f[30];
  float  seed = SEED,
   frand,
   valrand,
   epsilon = EPS,
   scale = SCALE,
   T,
   tt;
  float  far *v1;
  float  far *v2;


  switch(period1)
```

```
{
   case 1 :
     mk1 = MIL1;
     break;

   case 2 :
     mk1 = MIL2;
     break;
   case 3 :
     mk1 = MIL3;
     break;
   case 4 :
     mk1 = MIL4;
     break;
   case 5 :
     mk1 = MIL5;
     break;
   case 6 :
     mk1 = MIL6;
     break;
   default :
      fprintf(stderr,"\n Errore nella scelta in sign5\n");
      exit(1);

}

switch(period2)
```

```
{
   case 1 :
     mk2 = MIL1;
     break;
   case 2 :
     mk2 = MIL2;
     break;
   case 3 :
     mk2 = MIL3;
     break;
   case 4 :
     mk2 = MIL4;
     break;
   case 5 :
     mk2 = MIL5;
     break;
   case 6 :
     mk2 = MIL6;
     break;
   default :
      fprintf(stderr,"\n Errore nella scelta in sign5\n");
      exit(1);

}
  switch(period3)
{
   case 1 :
```

```
      mk3 = MIL1;
      break;
    case 2 :
      mk3 = MIL2;
      break;
    case 3 :
      mk3 = MIL3;
      break;
    case 4 :
      mk3 = MIL4;
      break;
    case 5 :
      mk3 = MIL5;
      break;
    case 6 :
      mk3 = MIL6;
      break;
    default :
       fprintf(stderr,"\n Errore nella scelta in sign5\n");
       exit(1);

}


switch(period4)
{
   case 1 :
```

```
    mk4 = MIL1;
    break;
  case 2 :
    mk4 = MIL2;
    break;
  case 3 :
    mk4 = MIL3;
    break;
  case 4 :
    mk4 = MIL4;
    break;
  case 5 :
    mk4 = MIL5;
    break;



  case 6 :
    mk4 = MIL6;
    break;
  default :
     fprintf(stderr,"\n Errore nella scelta in sign5\n");
     exit(1);

}

switch(period5)
```

```
 {
    case 1 :
      mk5 = MIL1;
      break;
    case 2 :
      mk5 = MIL2;
      break;
    case 3 :
      mk5 = MIL3;
      break;
    case 4 :
      mk5 = MIL4;
      break;
    case 5 :
      mk5 = MIL5;
      break;
    case 6 :
      mk5 = MIL6;
      break;
    default :
       fprintf(stderr,"\n Errore nella scelta in sign5\n");
       exit(1);

 }


freq1 = 1.0 / mk1;
```

```
 freq2 = 1.0 / mk2;
 freq3 = 1.0 / mk3;
 freq4 = 1.0 / mk4;
 freq5 = 1.0 / mk5;


fprintf(stderr,"\nIntervallo di campionamento : ");
scanf("%f",&T);
printf("\n");
punti = (int)(Np / T);

 v1 = (float far *) farcalloc(punti,sizeof(float));
 v2 = (float far *) farcalloc(punti,sizeof(float));

srand(seed);

/*----------------GENERAZIONE PRIMA E SECONDA SINUSOIDE----------------
*/

t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
  frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)           // ADD NOISE
{
```

```
 v1[t] = 1.0 + sin(2.0*pi*freq1*tt)  + valrand;
}

  else
       v1[t] = 1.0 + sin(2.0*pi*freq1*tt);

 frand = rand() / scale;
 valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)           // ADD NOISE
{
 v2[t] = 1.0 + sin(2.0*pi*freq2*tt)  + valrand;
}

  else
       v2[t] = 1.0 + sin(2.0*pi*freq2*tt);

  t += 1;
}

/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
```

```
    v1[t] = 0.2;


  if((v1[t] <= 1.53) && (v1[t] > 1.1))
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}

for(t=0;t < punti;t++)
{
  if(v2[t] < 0.67)
    v2[t] = 0.0;
  if((v2[t] <= 1.1) && (v2[t] >= 0.67))
    v2[t] = 0.2;
  if((v2[t] <= 1.53) && (v2[t] > 1.1))
    v2[t] = 0.4;
  if((v2[t] <= 1.96) && (v2[t] > 1.53))
    v2[t] = 0.6;
  if(v2[t] > 1.96)
    v2[t] = 0.8;
}

for(t=0;t < punti;t++)
 v2[t] += v1[t];
```

```
/*--------------------GENERAZIONE TERZA SINUSOIDE------------------*/

t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
  frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)            // ADD NOISE
{
 v1[t] = 1.0 + sin(2.0*pi*freq3*tt)  + valrand;
}

  else
       v1[t] = 1.0 + sin(2.0*pi*freq3*tt);

  t += 1;
}




/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
```

```
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}

for(t=0;t < punti;t++)
 v2[t] += v1[t];

/*----------------------GENERAZIONE QUARTA SINUSOIDE--------------------*/

t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
   frand = rand() / scale;
   valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)            // ADD NOISE
{
 v1[t] = 1.0 + sin(2.0*pi*freq4*tt)  + valrand;
}
```

```
  else
       v1[t] = 1.0 + sin(2.0*pi*freq4*tt);

  t += 1;
}

/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;

  if((v1[t] <= 1.53) && (v1[t] > 1.1))
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}

for(t=0;t < punti;t++)
 v2[t] += v1[t];
/*----------------------GENERAZIONE QUINTA SINUSOIDE------------------*/
```

```
t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
    frand = rand() / scale;
    valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)           // ADD NOISE
{
 v1[t] = 1.0 + sin(2.0*pi*freq5*tt)  + valrand;
}

  else
       v1[t] = 1.0 + sin(2.0*pi*freq5*tt);

  t += 1;
}

/*----------------------QUADRATURA-------------------------------------*/

for(t=0;t < punti;t++)
{
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
```

```
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}


for(t=0;t < punti;t++)              //  SOMMA FINALE
 v2[t] += v1[t];

for(t=0;t < punti;t++)              //  NORMALIZZAZIONE FINALE
{
  if(v2[t] <= 0.6)
    v2[t] = 0.0;
  if((v2[t] <= 1.4) && (v2[t] > 0.6))
    v2[t] = 0.2;
  if((v2[t] <= 2.2) && (v2[t] > 1.4))
    v2[t] = 0.4;
  if((v2[t] <= 3.0) && (v2[t] > 2.2))
    v2[t] = 0.6;
  if(v2[t] > 3.0)
    v2[t] = 0.8;
}


printf("\nnome file out ? ");
```

```
scanf("%s",f);
if((fout = fopen(f,"w")) == NULL)
 {
  fprintf(stderr,"\nimpossibile aprire il file %s\n",f);
  exit(1);
 }

for(t=0;t < punti;t++)
  fprintf(fout,"%f\n",v2[t]);

fclose(fout);

farfree(v1);
farfree(v2);
}

                              // SIGN6

void sign6(long Np)
{
  double mk1,mk2,mk3,mk4,mk5,mk6,freq1,freq2,freq3,freq4,freq5,freq6;
  int t,punti;
  FILE *fout;
  char f[30];
  float  seed = SEED,
   frand,
   valrand,
```

```
  epsilon = EPS,
  scale = SCALE,
  T,
  tt;
 float  far *v1;
 float  far *v2;


mk1 = MIL1;
mk2 = MIL2;
mk3 = MIL3;
mk4 = MIL4;
mk5 = MIL5;
mk6 = MIL6;


freq1 = 1.0 / mk1;
freq2 = 1.0 / mk2;
freq3 = 1.0 / mk3;
freq4 = 1.0 / mk4;
freq5 = 1.0 / mk5;
freq6 = 1.0 / mk6;


fprintf(stderr,"\nIntervallo di campionamento : ");
scanf("%f",&T);
printf("\n");
```

```
punti = (int)(Np / T);

v1 = (float far *) farcalloc(punti,sizeof(float));
v2 = (float far *) farcalloc(punti,sizeof(float));

srand(seed);

/*-----------------GENERAZIONE PRIMA E SECONDA SINUSOIDE-------------*/

t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
  frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)           // ADD NOISE
{
 v1[t] = 1.0 + sin(2.0*pi*freq1*tt) + valrand;
}

  else
       v1[t] = 1.0 + sin(2.0*pi*freq1*tt);

  frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));
if(frand > epsilon)           // ADD NOISE
```

```
{
 v2[t] = 1.0 + sin(2.0*pi*freq2*tt) + valrand;
}

  else
       v2[t] = 1.0 + sin(2.0*pi*freq2*tt);

  t += 1;
}

/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}

for(t=0;t < punti;t++)
```

```
{
  if(v2[t] < 0.67)
    v2[t] = 0.0;
  if((v2[t] <= 1.1) && (v2[t] >= 0.67))
    v2[t] = 0.2;
  if((v2[t] <= 1.53) && (v2[t] > 1.1))
    v2[t] = 0.4;
  if((v2[t] <= 1.96) && (v2[t] > 1.53))
    v2[t] = 0.6;
  if(v2[t] > 1.96)
    v2[t] = 0.8;
}

for(t=0;t < punti;t++)
 v2[t] += v1[t];

/*----------------------GENERAZIONE TERZA SINUSOIDE----------------------*/

t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
   frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)            // ADD NOISE
{
 v1[t] = 1.0 + sin(2.0*pi*freq3*tt)  + valrand;
```

```
}

  else
       v1[t] = 1.0 + sin(2.0*pi*freq3*tt);

  t += 1;
}

/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}

for(t=0;t < punti;t++)
 v2[t] += v1[t];
/*---------------------GENERAZIONE QUARTA SINUSOIDE-------------------*/
```

```
t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
   frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)            // ADD NOISE
{
 v1[t] = 1.0 + sin(2.0*pi*freq4*tt)  + valrand;
}

  else
       v1[t] = 1.0 + sin(2.0*pi*freq4*tt);

  t += 1;
}

/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
```

```
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}

for(t=0;t < punti;t++)
 v2[t] += v1[t];

/*----------------------GENERAZIONE QUINTA SINUSOIDE------------------*/

t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
   frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));

if(frand > epsilon)           // ADD NOISE
{
 v1[t] = 1.0 + sin(2.0*pi*freq5*tt)  + valrand;
}

  else
       v1[t] = 1.0 + sin(2.0*pi*freq5*tt);
  t += 1;
}
```

```
/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
   if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}

for(t=0;t < punti;t++)
 v2[t] += v1[t];

/*----------------------GENERAZIONE SESTA SINUSOIDE------------------*/

t = 0;
for(tt=0.0;tt < Np;tt+=T)
{
   frand = rand() / scale;
  valrand = (1.0/sqrt(2.0*3.14159)) * exp(-(frand*frand/2.0));
```

```
if(frand > epsilon)            // ADD NOISE
{
 v1[t] = 1.0 + sin(2.0*pi*freq6*tt)  + valrand;
}

  else
       v1[t] = 1.0 + sin(2.0*pi*freq6*tt);

  t += 1;
}

/*----------------------QUADRATURA-----------------------------------*/

for(t=0;t < punti;t++)
{
  if(v1[t] < 0.67)
    v1[t] = 0.0;
  if((v1[t] <= 1.1) && (v1[t] >= 0.67))
    v1[t] = 0.2;
  if((v1[t] <= 1.53) && (v1[t] > 1.1))
    v1[t] = 0.4;
  if((v1[t] <= 1.96) && (v1[t] > 1.53))
    v1[t] = 0.6;
  if(v1[t] > 1.96)
    v1[t] = 0.8;
}
```

```
for(t=0;t < punti;t++)              //  SOMMA FINALE
 v2[t] += v1[t];

for(t=0;t < punti;t++)              //  NORMALIZZAZIONE FINALE
{
  if(v2[t] < 0.8)
    v2[t] = 0.0;
  if((v2[t] <= 1.8) && (v2[t] >= 0.8))
    v2[t] = 0.2;
  if((v2[t] <= 2.8) && (v2[t] > 1.8))
    v2[t] = 0.4;
  if((v2[t] < 3.8) && (v2[t] > 2.8))
    v2[t] = 0.6;
  if(v2[t] >= 3.8)
    v2[t] = 0.8;
}

printf("\nnome file out ? ");
scanf("%s",f);
if((fout = fopen(f,"w")) == NULL)
 {
  fprintf(stderr,"\nimpossibile aprire il file %s\n",f);
  exit(1);
 }

for(t=0;t < punti;t++)
```

```
  fprintf(fout,"%f\n",v2[t]);

fclose(fout);

farfree(v1);
farfree(v2);
}
```

APPENDICE G

CODICE RELATIVO AL PROGRAMMA DI COSTRUZIONE DEI PATTERNS PER LE FASI DI TRAINING E DI TEST PER IL RICONOSCIMENTO DEI CICLI DI MILANKOVITCH.

```
/*************************************************************************
*
*          SORGENTE X COSTRUZIONE FRAMES INPUT
*          PER IL RICONOSCIMENTO DI SEGNALI DI MILANKOVITCH
*
*          RELEASE AT 08-05-1994
*
*                                        MAX BRESCIA
*
*************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <alloc.h>
#include <conio.h>
#include <string.h>

 #define   MIL1   ((float)410000)
 #define   MIL2   ((float)100000)
```

```
 #define   MIL3   ((float)49367)
 #define   MIL4   ((float)38202)
 #define   MIL5   ((float)22056)
 #define   MIL6   ((float)18346)


void main()
{
  double x,
    y,
    val1,
    val2,
    w,
    pi = 3.141592654;

    int N,
        n,
        k,
        Nd,
        i,
        incremento,
        inizio,
        fine,
        frame,
        answer,
        vartemp,
        index[6];
```

```
 float overlap,
 TOT,
 tot,
 varianza,
 fc,
 norm,
 min,
 max,
 outframe[6],
 target[6];

 FILE *fd,
*ff,
*fo;

 float far *v,
     *T,
     *f;

 char file_dati[30],
file_frame[30],
file_out[30];

 clrscr();

 fprintf(stderr,"\nnumero totale di punti ? ");
 scanf("%d",&N);
```

```
fprintf(stderr,"\nnumero dati per ogni frame ? ");
scanf("%d",&Nd);
fprintf(stderr,"\npercentuale di overlap ? ");
scanf("%f",&overlap);
fprintf(stderr,"\nintervallo di campionamento ? ");
scanf("%f",&fc);
fprintf(stderr,"\nnome file dati ? ");
scanf("%s",file_dati);

overlap /= 100.0;
fc = 1.0 / fc;                    //  FREQUENZA DI CAMPIONAMENTO

v = (float far *) farcalloc(N,sizeof(float));
T = (float far *) farcalloc(Nd/2,sizeof(float));
f = (float far *) farcalloc(Nd/2,sizeof(float));

if((fd = fopen(file_dati,"r")) == NULL)
 {
   fprintf(stderr,"\nimpossibile aprire il file %s\n",file_dati);
   farfree(v);
   farfree(T);
   farfree(f);
   exit(1);
 }

for(i=0;i < N;i++)
 fscanf(fd,"%f",&v[i]);
```

```
   fclose(fd);

/*--------------SOTTRAZIONE VARIANZA DAL SEGNALE TOTALE----------------
----*/

TOT = 0.0;
tot = 0.0;

for(i=0;i < N;i++)
{
  TOT += (v[i] * v[i]);
  tot += v[i];
}

varianza = (TOT - ((tot * tot)/(float)N)) / (float)(N-1);

for(i=0;i < N;i++)
 v[i] -= varianza;

/*---------------------INIZIO ELABORAZIONE FRAMES----------------------*/

incremento = (int)(Nd * overlap);
inizio = 0;
frame = 1;
fine = Nd;

printf("\nnome file pattern finale ? ");
```

```
scanf("%s",file_out);

printf("\ninserisci i valori target :\n(1.0 se la frequenza e' presente ");
printf("0.0 altrimenti\nfrequenza 1 : ");
scanf("%f",&target[0]);
printf("\nfrequenza 2 : ");
scanf("%f",&target[1]);
printf("\nfrequenza 3 : ");
scanf("%f",&target[2]);
printf("\nfrequenza 4 : ");
scanf("%f",&target[3]);
printf("\nfrequenza 5 : ");
scanf("%f",&target[4]);
printf("\nfrequenza 6 : ");
scanf("%f",&target[5]);

fo = fopen(file_out,"w");

while((fine-1) <= N)                    // CICLO PRINCIPALE
{
  printf("\nnome file frame %d ? ",frame);
  scanf("%s",file_frame);

  ff = fopen(file_frame,"w");

/*---------------------TRASFORMATA DI FOURIER----------------------------*/
```

```
min = 1000.0;
max = 0.0;

printf("\n\nCalculating the Fourier transform\nPlease wait...");

for(k=1;k < (Nd/2);k++)                 // BANDA [0,1/2[
 {
   w = (2.0*pi*k)/(float)(Nd);
   val1 = 0.0;
   val2 = 0.0;

   for(n=inizio;n < fine;n++)
     {
 y = - sin(w*n);
 x = cos(w*n);
 val1 += x * v[n];
 val2 += y * v[n];
     }

   f[k] = (fc*k) / Nd;                   // ASCISSA DELLO SPETTRO
   T[k] = sqrt((val1*val1) + (val2*val2));    // ORDINATA DELLO SPETTRO

   if(T[k] <= min)     min = T[k];             // CALCOLO MIN E MAX DELLO
SPETTRO
   if(T[k] >= max)  max = T[k];
 }
printf("...DONE!\n");
```

```
        /*-----------NORMALIZZAZIONE TRA -1 E 1 DELLO SPETTRO----------
--*/

         norm = (min + max) / 2.0;

         for(k=1;k < (Nd/2);k++)
            T[k] = (T[k] - norm) / max;

        /*-----------RICERCA DELLE FREQUENZE DI MILANKOVITCH----------
---*/

           k=1;
           i=0;
           while(f[k] <= 0.000057)
            {
              printf("\ncurrent f[%d] = %f with T[%d] = %f - ok? (1/0)--> ",k,
                                                          f[k],k,T[k]);
              scanf("%d",&answer);
              if(answer == 1)
               {
           index[i] = k;
           i++;
               }
              k++;
            }
```

```
        for(i=0;i < 6;i++)              // ASSEGNAZIONE DEI VALORI DEL FRAME
CORRENTE
        {
         vartemp = index[i];
         outframe[i] = T[vartemp];
        }


         for(k=1;k < (Nd/2);k++)
          fprintf(ff,"\n%f %f",f[k],T[k]);

       fclose(ff);

      /*-----------------C0MPOSIZIONE FRAMES FINALI PER LA RETE----------------
*/

         for(k=0;k < 6;k++)
            fprintf(fo,"%f ",outframe[k]);
         for(i=0;i < 6;i++)
            fprintf(fo,"%f ",target[i]);
         fprintf(fo,"\n");
        frame++;
        inizio += incremento;
        fine += incremento;
      }

      fclose(fo);
```

```
farfree(v);
farfree(T);
farfree(f);
}
```

RINGRAZIAMENTI

Desidero ringraziare :

il Prof. Roberto Tagliaferri, dell'Universita' degli Studi di Salerno, per il suo prezioso apporto, sia in fase di coordinamento del lavoro, sia in fase di suggerimento dei modelli realizzati.

il Prof. Bruno D'Argenio, direttore dell'Istituto di Geologia Marina del CNR "GEOMARE SUD", di Napoli, per aver fornito tutte le nozioni necessarie di geologia e per i continui incoraggiamenti durante la realizzazione del lavoro.

il Dott. Nicola Pelosi, dell'Istituto "GEOMARE SUD" di Napoli, per l'alto grado di sopportazione della nostra continua presenza e per i validissimi suggerimenti tecnici.

il Dott. Giuseppe Longo, dell'Osservatorio Astronomico di Capodimonte, per averci fornito preziose nozioni tecniche e supporti bibliografici e per aver reso meno tedioso e piu' imprevedibile il lavoro, con le sue continue "sparizioni".

il Dott. Salvatore Rampone, dell'Universita' degli studi di Salerno, per le interminabili ore di preziosissima collaborazione prestataci, spesso risultata positivamente decisiva.

la Prof.ssa Maria Marinaro, preside della Facolta' di Scienze dell'Universita' degli Studi di Salerno e presidente dell'Istituto IIASS di Vietri sul Mare, per aver consentito il conseguimento della laurea e per averci permesso l'utilizzo delle apparecchiature elettroniche con cui eseguire gli esperimenti.

l'amico e collega Ciro D'Urzo per l'aiuto psicologico e materiale elargito durante tutta la prosecuzione del lavoro.

Inoltre, desidero rivolgere un ulteriore ringraziamento sia alle persone non nominate, che a quelle su menzionate, per la pazienza, la disponibilita' e la professionalita' rivoltaci, a testimonianza della totale generosita', imparzialita' ed assenza di pregiudizi presenti nella comunita' scientifica.

Massimo Brescia

BIBLIOGRAFIA

[1] A. V. Oppenheim R. W. Schafer, Elaborazione numerica dei segnali, Milano 1993, Franco Angeli Editore.

[2] Jack D. Gaskill, Linear Systems, Fourier Transforms and Optics, N.Y. 1978, John Wiley & Sons.

[3] John E. Freund, Mathematical Statistics, London 1985, Prentice Hall International.

[4] Nicola Pelosi Arturo Raspini, Analisi spettrale della ciclicita' di alta frequenza nella successione carbonatica cretacica dei monti di Sarno (Campania), Napoli 1993, Progetto Geomare, Istituto di Geologia Marina del CNR.

[5] B. D'Argenio V. Ferreri F. Ardillo F. P. Buonocunto N. Pelosi, Il cretacico superiore del monte Tobenna (Salerno); Studio sulla ciclicita' di alta frequenza in depositi carbonatici di piattaforma, Napoli 1993, Istituto Geomare Sud.

[6] Jeffrey D. Scargle, Studies in astronomical time series analysis; Statistical aspects of spectral analysis of unevenly spaced data, The Astrophysical Journal 263: 835 - 853, 1982 December 15.

[7] James H. Horne Sallie L. Baliunas, A prescription for period analysis of unevenly sampled time series, The Astrophysical Journal, 302: 757 - 763, 1986 March 15.

[8] Chris Koen, Significance testing of periodogram ordinates, The Astrophysical Journal, 348: 700 - 702, 1990 January 10.

[9] David J. Thomson, Spectrum Estimation and Harmonic Analysis, Proceedings of the IEEE, Vol. 70, N. 9, September 1982.

[10] Linda A. Hinnov Robert K. Goldhammer, Spectral Analysis of the middle triassic Latemar Limestone, Journal of Sedimentary Petrology, Vol. 61, N. 7, December 1991.

[11] Jeffrey Park Timothy D. Herbert, Hunting for paleoclimatic periodicities in a geologic time series with an uncertain time scale, Journal of Geophysical Research, Vol. 92, N. B13, December 1987.

[12] D. Slepian, Prolate spheroidal wave functions, Fourier analysis and uncertainty V: the discrete case, The Bell System Technical Journal, Vol. 57, N. 5, May-June 1978.

[13] L. Fortuna S. Graziani G. Nunnari, Le reti neurali per la modellistica e l'analisi dei segnali in Geofisica, Geoinformatica, Vol. 1, N. 0, Catania 1991.

[14] A. Waibel T. Hanazawa G. Hinton K. Shikano, Phoneme Recognition using Time-Delay neural networks, ATR Interpreting Telephony Research Lab., October 30, 1987.

[15] K. J. Lang A. Waibel G. E. Hinton, A Time-Delay neural netwok Architecture for isolated word recognition, Neural Networks, Vol. 3, 1990.

[16] T. Corner C. Leiserson R. Rivest, Introduction to Algorithms, MIT Press, McGraw Hill, 1990.

[17] J. L. McClelland D. E. Rumelhart, Parallel Distributed Processing, MIT Press, 1987.

[18] S. Cammarata, Reti Neuronali, Etas Libri, Milano 1990.

[19] R. C. Eberhart R. W. Dobbins, Neural Network PC tools, Academic Press, California 1990.

[20] L. Napolitano, Borland C++ & Application Frameworks, Jackson Libri, Milano 1992.

[21] S. C. Dewhurst K. T. Stark, Programmazione in C++, Prentice Hall Int., Milano 1989.

[22] Dott. N. Pelosi, Comunicazione Personale, 1994.

[23] Dott. B. D'Argenio, Comunicazione Personale, 1994.

[24] Dott. G. Nunnari, Comunicazione personale, 1994.

[25] Dott. G. Longo, OAC Napoli, Comunicazione Personale, 1994.

[26] A. Papoulis, Probability, Random variables and Stochastic processes, McGraw Hill, N.Y. 1965.

[27] A. Erdelyi, Higher trascendental functions, McGraw Hill, N.Y. 1953.

[28] C. J. Tranter, Bessel functions with some physical applications, The English Universities Press, London 1968.

[29] G. B. Whitham, Linear and nonlinear waves, J. Wiley and Sons, N.Y. 1974.

[30] H. S. Black, Modulation theory, Van Nostrand Reinhold Company, N. Y. 1953.

[31] K. Chakraborty et al., Forecasting the behavior of multivariate time series using neural networks, Neural Networks,Vol. 5: 961 - 970, 1992.

[32] P.C. Woodland, Weight limiting, weight quantisation & generalisation in multilayer perceptrons, Proc. First IEEE International Conf. on Artificial Neural Networks (Oct. 1989), 297-300.

[33] M.W. Mak, W.G. Allen, G. Sexton, Speaker identification using multilayer perceptrons networks, Neurocomputing, Vol. 6, Numero 1 Febbraio 94, 99-117.

[34] W.H. Press, B.P. Flannery, Numerical Recipes, Cambridge University Press, 1986.

INDICE

__oOo__