Sede Amministrativa: UNIVERSITÀ DEGLI STUDI DI PADOVA

Sede Consorziata: UNIVERSITÀ CA' FOSCARI DI VENEZIA

Dipartimento di Scienze Ambientali

Dottorato di Ricerca in:

Modellistica dei Sistemi Ambientali

XI ciclo

PROCESSI CAOTICI NELL'ATMOSFERA: UN MODELLO DI PREVISIONE DELLA TEMPERATURA DELL'ARIA

Coordinatore: Ch.mo Prof. Giovanni Marchesini

Supervisore: Ch.mo Prof. Alessandro Marani

Dottorando: Giovanni Dal Bo'

31 dicembre 1999

INDICE

Riassunto

Nel presente lavoro sono stati studiati e sviluppati i metodi di analisi dei sistemi caotici, utilizzando come dati di riferimento e controllo quelli generati dalle dinamiche caotiche più note in letteratura: funzione logistica e sistemi di Hénon e Lorenz.
L'applicazione di questi metodi ad una serie temporale di dati sperimentali (temperatura dell'aria al suolo a Udine-Castello) ha permesso di dimostrare che tale grandezza è governata, almeno localmente, da un sistema caotico a 5 gradi di libertà e non da un sistema stocastico.
Le informazioni sull'evoluzione temporale di un punto nello spazio delle fasi contenute nell'attrattore di un sistema caotico sono state utilizzate per sviluppare un modello locale di previsione.
Tale modello è stato applicato ai dati di temperatura ed ha fornito previsioni abbastanza buone, almeno entro un limite pari a tre volte il periodo di campionamento della serie in esame.

Abstract

The techniques of analysis of chaotic systems have been studied and developed. Data generated from most known chaotic dynamics (logistics function , Hénon and Lorenz systems) have been used to check the outputs of the programs.
The application of these methods to a time-series of experimental data (ground air temperature at Udine-Castello) has allowed to show that this variable is governed, at least locally, by a chaotic system with 5 degrees of freedom and not by a stocastic system.
The information on the temporal evolution of a point in the space of phases contained in the attractor of a chaotic system have been used to develop a local forecasting model.
This model has been applied to the temperature data and it has provided fairly good forecasts, at least within a limit of three times the period of sampling of the time-series.

1. Introduzione

La comprensione di un fenomeno naturale si conclude con la determinazione di un insieme di equazioni capaci di descrivere la dinamica del sistema. A tale risultato si perviene generalmente attraverso una serie di azioni quali: l'identificazione del fenomeno; una serie di esperimenti finalizzati alla raccolta dei dati; un numero adeguato di simulazioni; analisi statistiche dei dati sperimentali e simulati. Spesso, soprattutto nel caso dei sistemi ambientali, l'analisi dei dati fa sorgere la questione se ci sia una qualche dinamica deterministica che governa il fenomeno oppure se i dati non siano altro che numeri generati da un sistema stocastico senza alcuna struttura soggiacente. In effetti in un universo macroscopico retto *comunqu*e da leggi deterministiche, un sistema stocastico è un sistema con un numero così grande di gradi di libertà e il cui comportamento è determinato da un così elevato numero di equazioni che risulta impossibile darne una formulazione matematica anche approssimativa. Un tale sistema può essere descritto solo con metodi statistici. Viceversa, nel caso di un sistema a basso numero di gradi di libertà è possibile, almeno in linea di principio, ricavare le equazioni della dinamica, e ottenere così una comprensione accurata del fenomeno.

Dal punto di vista dell'osservatore, un processo può essere considerato come una scatola nera contenente un sistema dinamico continuo che si manifesta attraverso la misura di una sua variabile effettuata ad intervalli di tempo discreti. Nel caso di

misurazioni regolari il *segnale* da interpretare è una *serie temporale* di dati ottenuti campionando il sistema ad intervalli di tempo τ :

$$s(n) = s(t_0 + n\tau), \qquad n = 0,1,2,3\ldots \tag{1.1}$$

A prima vista, la discriminazione tra il segnale generato da un sistema stocastico e quello generato da un sistema dinamico a pochi gradi di libertà non sembra un problema difficile: un semplice esame del segnale stesso o al massimo un'analisi statistica classica dovrebbero consentire di distinguere i due casi. In realtà utilizzando strumenti convenzionali lineari quali ad esempio l'analisi di Fourier, tale discriminazione risulta spesso quanto meno problematica.

Si considerino ad esempio le serie temporali di Figura 1.1, ciascuna con il relativo spettro di potenza: i due segnali sembrano entrambi casuali e non dotati di

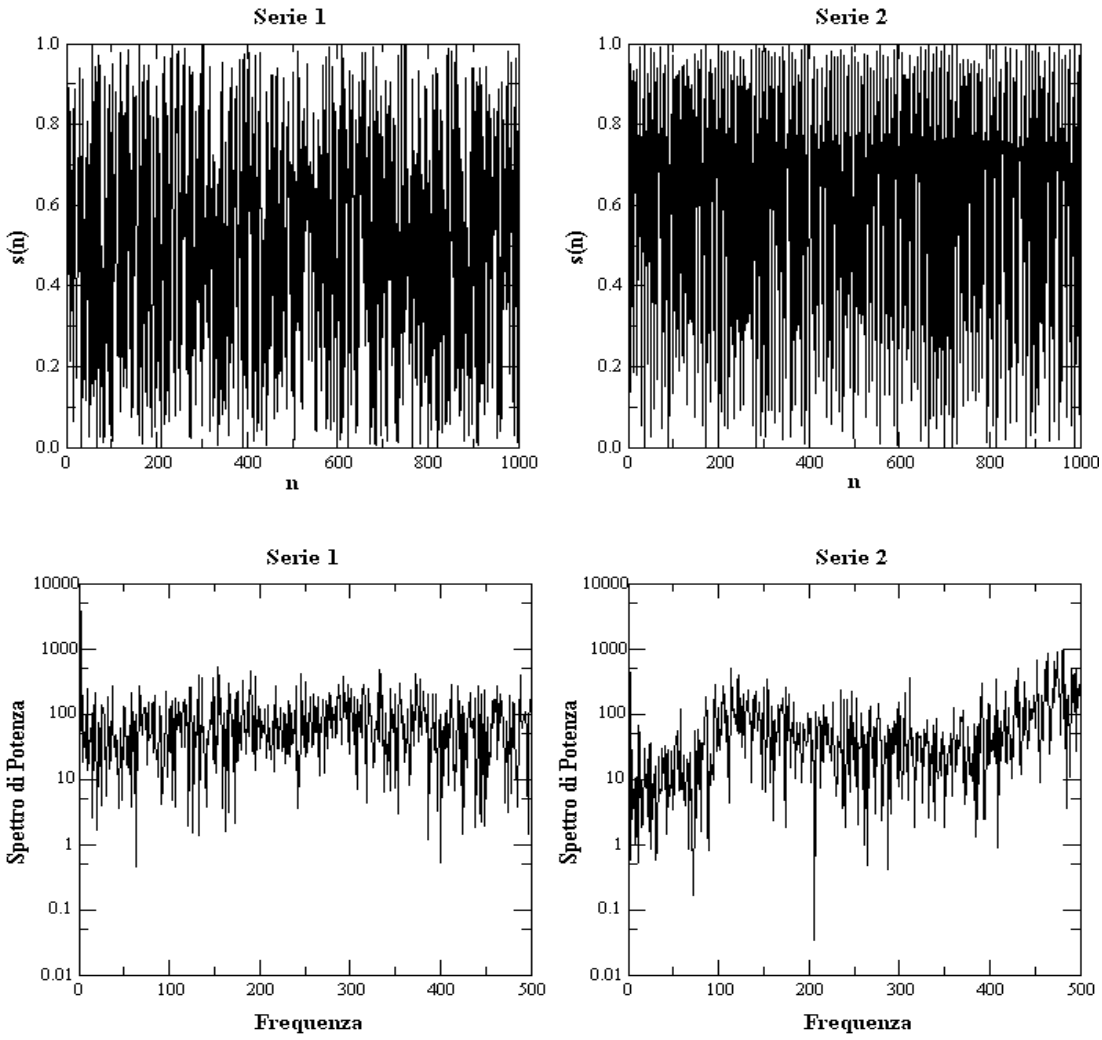

FIGURA 1.1 Due serie temporali (in alto) ed i relativi spettri di potenza (in basso).

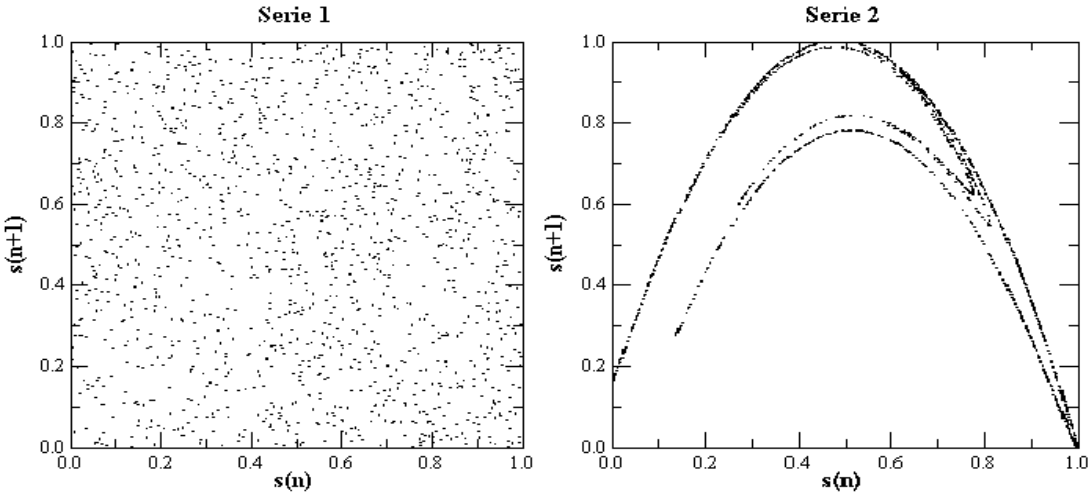

FIGURA 1.2 $s(n+1)$ in funzione di $s(n)$ per le due serie di Figura 1.1.

una qualche frequenza caratteristica. Se si rappresenta però il dato $s(n+1)$ in funzione di $s(n)$ (Figura 1.2), emerge una differenza significativa: nel secondo caso è chiaramente presente una struttura ad indicare che il segnale non è prodotto da un processo stocastico.

1.1 Il piano della tesi

Nel presente capitolo vengono introdotti i sistemi caotici e le loro principali caratteristiche; segue poi una breve discussione sul caos dell'atmosfera.

Il secondo capitolo verte sull'analisi dei sistemi caotici con particolare riferimento alla ricostruzione dello spazio delle fasi.

Nel terzo capitolo saranno discussi gli invarianti della dinamica caotica con particolare riguardo per la dimensione frattale dell'attrattore e per gli esponenti di Lyapunov.

Nel quarto capitolo verrà descritta la serie storica di temperature registrate a Udine, gentilmente fornita dal prof. Mario Ceschia dell'Università di Udine, e utilizzata nel presente lavoro; verranno inoltre illustrate le tecniche di *denoising* della serie.

Nel quinto capitolo l'applicazione delle le tecniche di analisi caotica consentirà di dimostrare che la serie storica in esame è generata da un sistema caotico

a basso numero di gradi di libertà. Tale conclusione è accompagnata ed avvalorata da un modello concettualmente molto semplice che, basandosi sulla struttura dell'attrattore del sistema, è in grado di effettuare delle buone previsioni della variabile considerata.

1.2 Sistemi caotici

Con il termine *caos* viene indicata una classe di segnali che hanno un comportamento intermedio tra un andamento regolare, periodico o quasiperiodico, ed un andamento stocastico e del tutto imprevedibile.

In generale, l'evoluzione di un sistema caotico è determinata da un sistema di equazioni differenziali non lineari:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{F}(\mathbf{x}(t)) \tag{1.2}$$

con tre o più gradi di libertà o da una mappa discreta invertibile:

$$\mathbf{x}(t+1) = \mathbf{F}(\mathbf{x}(t)) \tag{1.3}$$

con due o più gradi di libertà (Thompson and Stewart, 1986)[(1)]. Il numero di gradi di libertà è rappresentato dal numero di equazioni differenziali ordinarie del primo ordine necessarie a descrivere l'evoluzione in un sistema continuo ovvero dal numero di componenti del vettore di stato $\mathbf{x}(t)$. Le orbite descritte dal punto $\mathbf{x}(t)$ di un sistema caotico sono caratterizzate da un'elevata complessità, dall'assenza di periodicità e dalla dipendenza sensibile (esponenziale) dalle condizioni iniziali. Al crescere di *t* tali orbite tendono ad occupare un sottoinsieme dello spazio delle fasi detto *attrattore*.

A differenza dei sistemi regolari, per i quali l'attrattore è un punto fisso, un'orbita periodica (ciclo limite) o un'orbita quasi-periodica (toro) e la cui dimensione è un numero intero, nel caso di un sistema caotico si parla di *attrattore*

[1] In effetti la funzione logistica (cfr. par. 1.2.1.) è un sistema caotico la cui evoluzione è descritta da una mappa discreta monodimensionale.

strano in quanto caratterizzato da una dimensione non intera (oggetto frattale) e dal fatto che traiettorie passanti da punti anche molto vicini finiscono per divergere.

1.2.1 La funzione logistica

Un sistema dinamico può avere comportamenti regolari per particolari valori dei parametri delle equazioni che lo descrivono e comportamenti caotici per altri. Un tipico esempio è quello della funzione logistica o iteratore quadratico:

$$x_{k+1} = r x_k (1 - x_k) \tag{1.4}$$

la cui origine risale ad un modello di dinamica della popolazione dovuto al matematico belga Pierre François Verhulst.

Se si rappresenta in funzione del parametro (che rappresenta il tasso di crescita della popolazione) l'evoluzione asintotica del sistema, x_∞, si vede come, al crescere di *r*, il sistema compia una transizione da stabile, con uno o più punti fissi, a caotico (Figura 1.3).

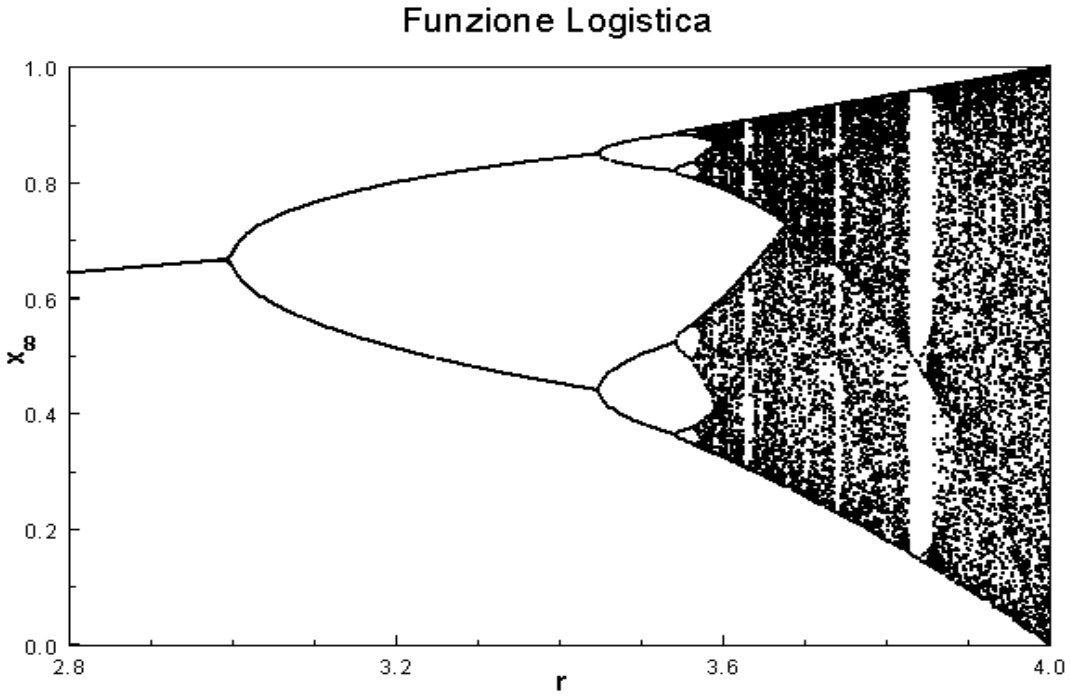

FIGURA 1.3 Diagramma dell'evoluzione asintotica della funzione logistica per valori del parametro compresi tra 2.8 e 4. Il valore iniziale è $x_0 = 0.8$.

1.2.2. Il sistema di Lorenz

Un celebre esempio di sistema caotico è il modello derivato da Lorenz nel 1963 per approssimare la convezione termica nella bassa atmosfera:

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = -xz + Rx - y \\ \dot{z} = xy - Bz \end{cases} \tag{1.5}$$

ove le grandezze σ, B e R sono i parametri del sistema che Lorenz nei suoi studi originali aveva fissato ai valori:

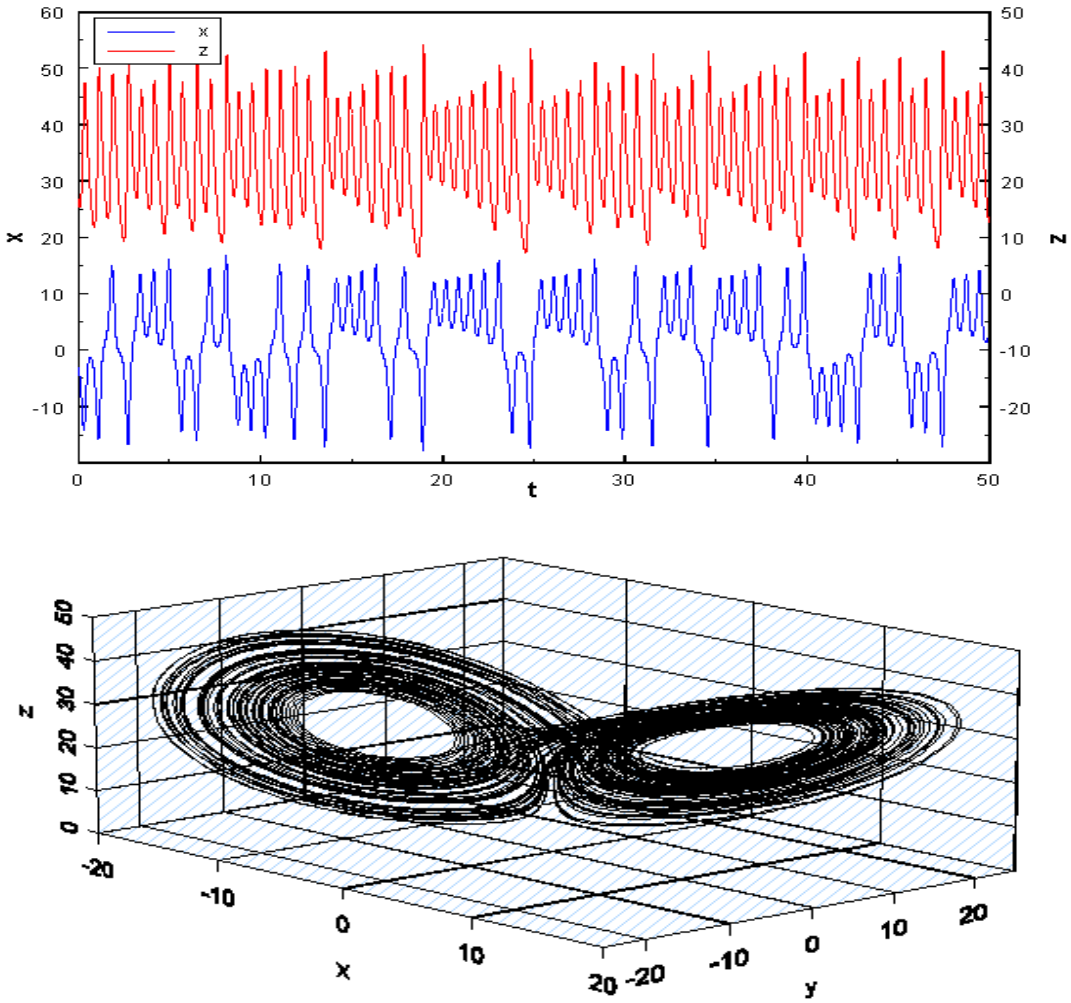

FIGURA 1.4 Serie di valori delle variabili x e z (in alto) e attrattore del sistema di Lorenz (in basso). I valori dei parametri sono quelli fissati in origine da Lorenz: $\sigma = 10$, $B = 8/3$ e $R = 28$; il passo d'integrazione è: $\tau = 0.005$.

$$\sigma = 10 \ , \quad B = \frac{8}{3} \ , \quad R = 28 \, .$$

Le serie di valori delle variabili x e z e l'attrattore del sistema di Lorenz sono rappresentati in Figura 1.4.

Anche nel caso del sistema di Lorenz, per opportuni valori dei parametri le equazioni possono ammettere soluzioni stabili come mostra la Figura 1.5 dove sono rappresentate le serie di valori delle variabili x e z (in alto) e il ciclo limite del sistema (in basso) che corrispondono alla soluzione stabile che si ottiene quando i parametri σ, B ed R valgono rispettivamente 10, 8/3 e 100.

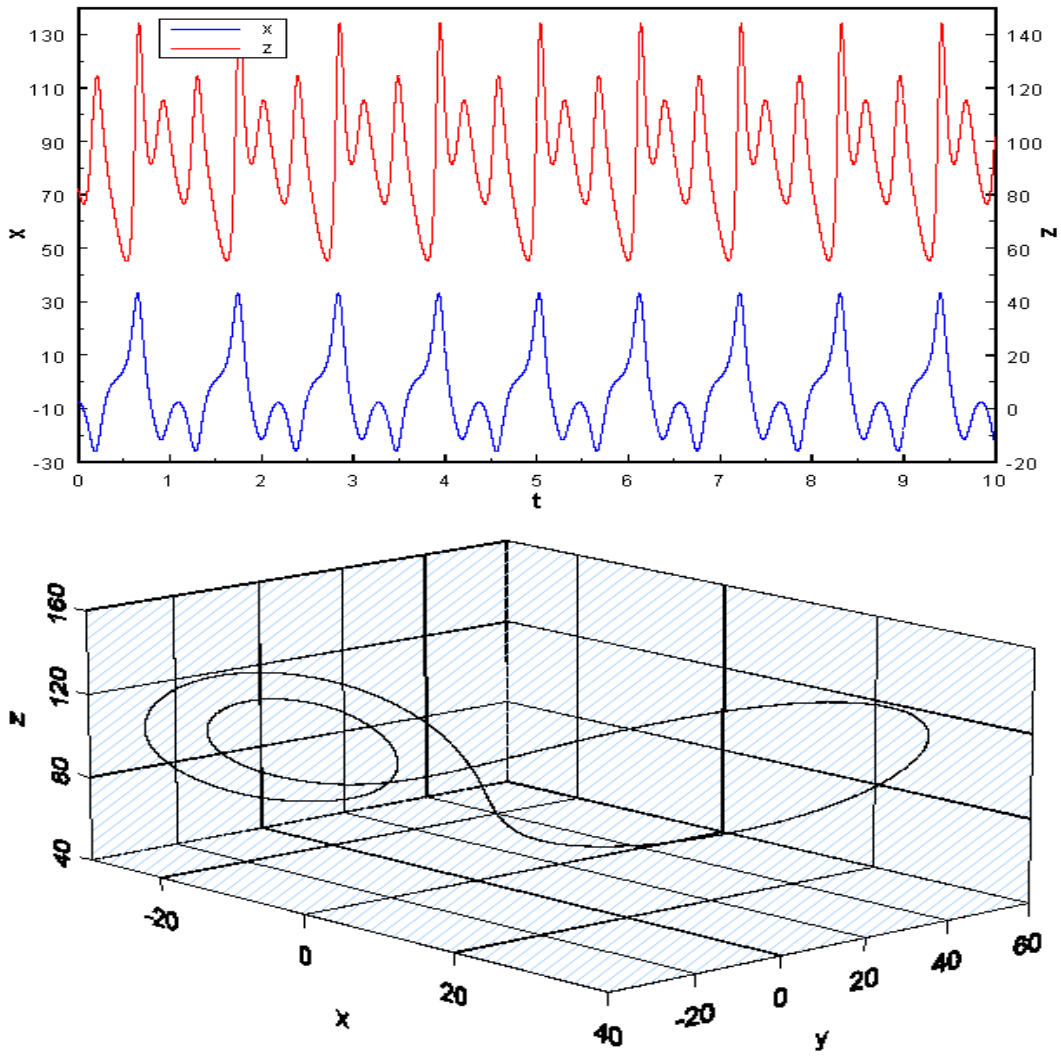

FIGURA 1.5 Serie di valori delle variabili x e z (in alto) e ciclo limite del sistema di Lorenz (in basso). I valori dei parametri sono: $\sigma = 10$, $B = 8/3$ e $R = 100.795$; il passo d'integrazione è: $\tau = 0.005$.

1.2.3. Il sistema di Hénon

Nel 1976 l'astronomo francese Michel Hénon propose un modello semplificato della dinamica del sistema di Lorenz, definito dalla seguente mappa discreta bidimensionale (Hénon, 1976):

$$\begin{cases} x_{k+1} = -ax_k^2 + y_k + 1 \\ y_{k+1} = bx_k \end{cases}, \qquad k = 0,1,2,\ldots \tag{1.6}$$

ove i valori dei parametri scelti da Hénon sono:

$$a = 1.4 \quad \text{e} \quad b = 0.3$$

L'attrattore del sistema di Hénon è rappresentato in Figura 1.6.

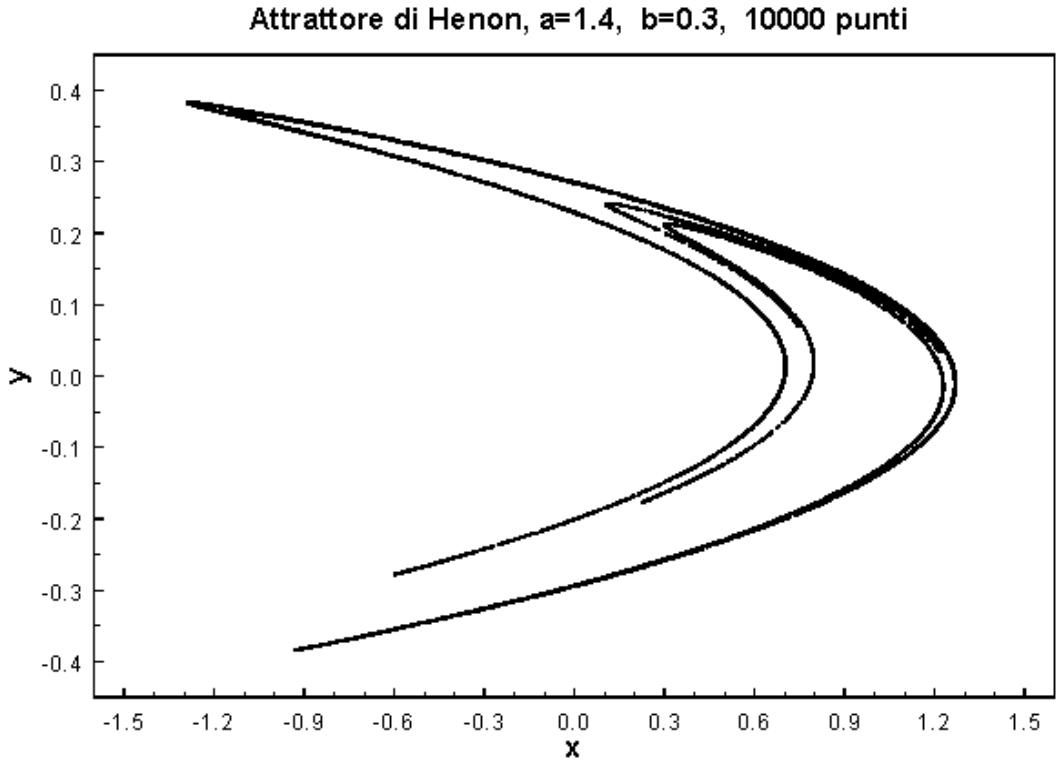

FIGURA 1.6 Attrattore del sistema di Hénon (1.6).

1.2.4 Imprevedibilità dei sistemi caotici

Una caratteristica dei sistemi caotici è l'imprevedibilità derivante dal fatto che traiettorie passanti per punti anche molto vicini finiscono per divergere.

In un sistema caotico la distanza tra le orbite passanti per due punti dello spazio delle fasi arbitrariamente vicini aumenta esponenzialmente col tempo fino a divenire confrontabile con l'ampiezza dell'attrattore stesso, dopodiché le due orbite sono completamente scorrelate e la loro distanza assume un andamento casuale. Tale fenomeno è noto anche come *dipendenza sensibile dalle condizioni iniziali*. La Figura 1.7 mostra l'evoluzione nel tempo della distanza tra due punti inizialmente molto vicini della funzione logistica (1.4).

Come vedremo più avanti, la divergenza media delle traiettorie, e quindi l'imprevedibilità di un sistema, può essere quantificata mediante gli esponenti di Lyapunov.

Un'altra grandezza che caratterizza un sistema caotico è la *dimensione frattale* del suo attrattore. La relazione che esprime come un (iper) volume scala in

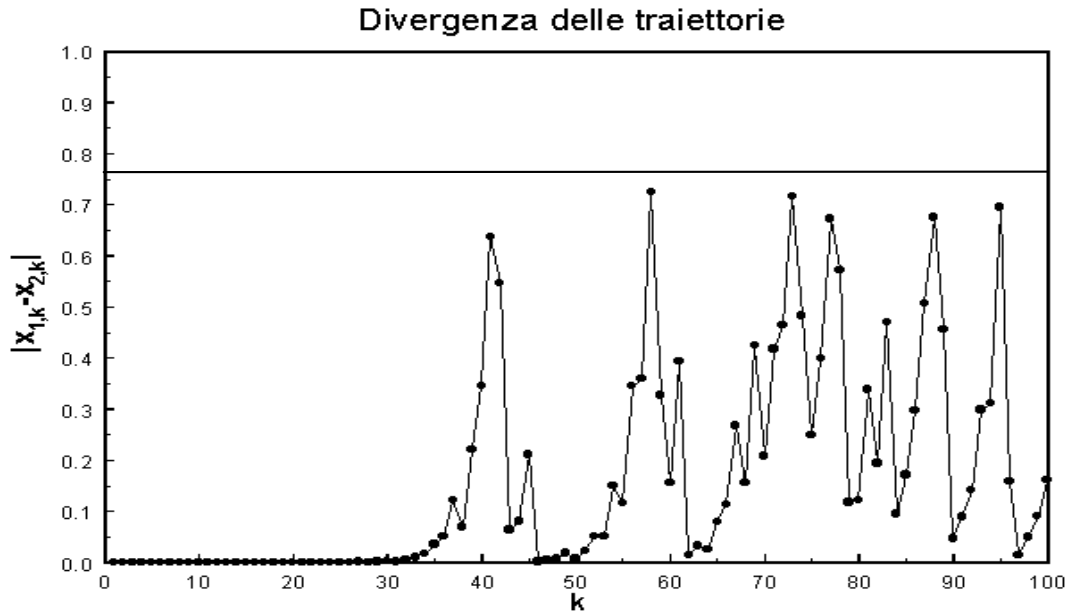

FIGURA 1.7 Evoluzione della distanza tra i punti $x_{1,0} = 0.1$ e $x_{2,0} = 0.10000001$ della funzione logistica (1.4). Il parametro *r* vale 3.8; in corrispondenza di tale valore l'attrattore è l'intervallo di estremi 0.18 e 0.95, la cui ampiezza, pari a 0.77, è rappresentata dalla linea orizzontale.

funzione della lunghezza:

$$V \propto L^D$$

può essere invertita per "definire" la dimensione:

$$D = \frac{\log V}{\log L}. \qquad (1.7)$$

Vedremo in seguito come da una generalizzazione della nozione di volume nella (1.7) derivino le diverse definizioni di dimensione frattale ed i relativi metodi per ottenere una stima numerica di tali grandezze.

1.3 Il caos dell'atmosfera

Se l'atmosfera sia un sistema caotico è una questione che è stata a lungo dibattuta. Tale dibattito inizia nel 1963 con la pubblicazione dell'ormai famoso modello di Lorenz che viene spesso menzionato come il primo esempio di sistema caotico.

Un lavoro di Nicolis pubblicato su *Nature* (Nicolis, 1984) pone la questione dell'esistenza o meno di un attrattore per il clima del passato: applicando l'algoritmo di Grassberger-Procaccia(2) ad una serie temporale di circa 500 punti, egli trova una dimensione dell'attrattore strano pari a 3.1.

Il risultato di Nicolis fu tuttavia immediatamente contestato dallo stesso Grassberger, il quale utilizzando 7000 punti stabilì che la dimensione era maggiore di 10.

Il dibattito continuò negli anni successivi (Grassberger, 1986; Nicolis, 1987; Essex, 1987; Tsonis, 1988)

Tra gli articoli più interessanti sull'argomento, ricordiamo quello di Zeng et al. (1991), relativo all'analisi per mezzo degli esponenti di Lyapunov di serie di temperature in due località degli Stati Uniti al fine di valutare quantitativamente la prevedibilità di tali grandezze. Dallo studio emerge in particolare che le serie delle

[2] Cfr. par. 3.2.

temperature hanno due esponenti di Lyapunov positivi, pertanto l'atmosfera ha un attrattore caotico con un tempo di raddoppio dell'errore di circa 3.7 giorni in una delle due località considerate (Los Angeles) dove il rapporto segnale/rumore è alto, e di circa 2.5 giorni a Fort Collins (l'altra località esaminata) dove il rapporto segnale/rumore è più basso.

Si rimanda infine ai riferimenti bibliografici per un elenco dei principali articoli che si trovano in letteratura sull'argomento.

2. Analisi di un sistema caotico

2.1 Ricostruzione dello spazio delle fasi

Il primo problema da risolvere nell'analisi di un sistema caotico è la ricostruzione dello spazio delle fasi a partire da un segnale monodimensionale. La risposta a questo problema è fornita dal seguente teorema di *embedding* (Mañé et al., 1981; Takens, 1981):

"Data la grandezza scalare h funzione di una grandezza vettoriale $\mathbf{g}$ a sua volta funzione delle variabili di stato $\mathbf{x}$:

$$h = h(\mathbf{g}(\mathbf{x}(n))), \tag{2.1}$$

lo spazio composto dai vettori $\mathbf{y}$ le cui componenti sono date da h applicata a potenze di $\mathbf{g}(\mathbf{x}(n))$ [3]:

$$\mathbf{y}(n) = [h(\mathbf{x}(n)), h(\mathbf{g}^{T_1}(\mathbf{x}(n)), h(\mathbf{g}^{T_2}(\mathbf{x}(n)), \ldots, h(\mathbf{g}^{T_{d-1}}(\mathbf{x}(n))] \tag{2.2}$$

---

[3] Si indica come potenza T di una funzione $\mathbf{g}$ la applicazione ripetuta T volte della funzione stessa: $\mathbf{g}^T(x) = \mathbf{g}(\mathbf{g}(\cdots(\mathbf{g}(x))\cdots))$, cosicché $\mathbf{g}^0(x) = x$, $\mathbf{g}^1(x) = \mathbf{g}(x)$, $\mathbf{g}^2(x) = \mathbf{g}(\mathbf{g}(x))$, …

è tale da riprodurre senza ambiguità le proprietà del segnale multivariato incognito $\mathbf{x}(n)$".

In particolare l'evoluzione nel tempo dei punti $\mathbf{y}$ segue quella della dinamica incognita $\mathbf{x}$ nel senso che il comportamento deterministico della dinamica $\mathbf{x}$ assicura il comportamento deterministico della dinamica sostitutiva $\mathbf{y}$ (Abarbanel, 1995).

Se la funzione h rappresenta la serie dei valori osservati: $h(\mathbf{x}(n)) = s(n)$ e se la funzione $\mathbf{g}(\mathbf{x})$ trasla in avanti il vettore $\mathbf{x}$ di un tempo pari a quello di campionamento τ: $\mathbf{g}(\mathbf{x}(n)) = \mathbf{x}(n+\tau)$, allora la (2.2) diviene:

$$\mathbf{y}(n) = [s(n), s(n+\tau T_1), s(n+\tau T_2), \ldots, s(n+\tau T_{d-1})] \qquad (2.3)$$

posto inoltre $\tau T_k = kT$, si ha:

$$\mathbf{y}(n) = [s(n), s(n+T), s(n+2T), \ldots, s(n+(d-1)T)]. \qquad (2.4)$$

Fissati il *time delay* T e la dimensione di *embedding* d, i vettori $\mathbf{y}(n)$ rappresentano un sostituto dei vettori di stato $\mathbf{x}(t)$ del sistema dinamico originale. Tale rappresentazione è "fedele" nel senso che conserva gli invarianti della dinamica originale senza perdita di informazioni. È quindi di fondamentale importanza poter disporre di un criterio – o quanto meno, di una prassi – che consenta di determinare il *time delay* e la dimensione di *embedding*.

2.2 Time delay

Il teorema di *embedding* non pone restrizioni sul *time delay* che, in linea di principio, può assumere un valore qualsiasi. Tale grandezza deve comunque, quanto meno, soddisfare alcuni requisiti:

1) poiché si suppone che il segnale sia campionato con periodo τ, il *time delay* T dovrà essere un multiplo di τ;
2) il *time delay* non deve essere troppo breve affinché le componenti del vettore $\mathbf{y}(n)$ siano sufficientemente indipendenti. Infatti, se T fosse molto piccolo

rispetto alla scala temporale della dinamica osservata, le componenti $s(n)$ ed $s(n+T)$ conterrebbero sostanzialmente la stessa informazione;

3) il *time delay* non deve essere troppo lungo affinché le componenti del vettore $\mathbf{y}(n)$ non siano del tutto indipendenti. A causa della divergenza esponenziale delle traiettorie di un sistema caotico, se T fosse molto grande rispetto alla scala temporale della dinamica osservata, le componenti $s(n)$ ed $s(n+T)$ sarebbero random una rispetto all'altra;

Sulla base di questi requisiti, diversi criteri sono stati individuati per determinare il valore ottimale del *time delay* e, sebbene non ne esista uno universalmente accettato (Rosenstein et al., 1994), due sono i metodi più diffusi in letteratura – per un terzo criterio di determinazione del *time delay* si veda ad es. (Venkadesan et al., 1996).
Il primo criterio è basato sulla funzione di autocorrelazione lineare:

$$C_L(T) = \frac{\frac{1}{N}\sum_n [s(n+T)-\bar{s}][s(n)-\bar{s}]}{\frac{1}{N}\sum_n [s(n)-\bar{s}]^2} \qquad (2.5)$$

dove

$$\bar{s} = \frac{1}{N}\sum_{n=1}^{N} s(n)\,.$$

In base a questo criterio, il *time delay* ottimale è quello in corrispondenza del quale la funzione di autocorrelazione lineare (2.5) ha il primo zero (Abarbanel et al., 1993), oppure si riduce di un fattore pari a $1/e$ (*e-fold time*) (Rosenstein et al., 1993).

Tale scelta è stata però oggetto di critica (Abarbanel, 1995) proprio perché basata sull'uso dell'autocorrelazione lineare giudicato improprio nel caso di sistemi non lineari quali quelli caotici.

Un altro criterio per la determinazione di T (Fraser and Swinney, 1986) è basato sul concetto di *mutua informazione* tra la misura a_i appartenente all'insieme $A=\{a_i\}$ e la misura b_j appartenente all'insieme $B=\{b_j\}$:

$$MI(a_i b_j) = \log_2 \left[\frac{P_{AB}(a_i, b_j)}{P_A(a_i) P_B(b_j)} \right] \qquad (2.6)$$

dove $P_{AB}(a,b)$ è la densità di probabilità congiunta che le misure di *A* e *B* diano per risultato *a* e *b* rispettivamente; $P_A(a)$ e $P_B(b)$ sono le densità di probabilità che la misura di *A* dia *a* e quella di *B* dia *b*. Se il risultato a_i, ottenuto misurando *A*, è del tutto indipendente dal fatto che si sia ottenuto b_j misurando *B*, allora $P_{AB}(a_i, b_j) = P_A(a_i) P_B(b_j)$ e la mutua informazione è zero. La *mutua informazione media* tra le misure di A e quelle di B è data da:

$$I_{AB} = \sum_{a_i b_j} P_{AB}(a_i, b_j) MI(a_i b_j) . \qquad (2.7)$$

Se si pone: $a_i = s(n)$ e $b_j = s(n+T)$, la (2.7) diviene:

$$I(T) = \sum_{s(n), s(n+T)} P(s(n), s(n+T)) \log_2 \left[\frac{P(s(n), s(n+T))}{P(s(n)) P(s(n+T))} \right] \qquad (2.8)$$

La (2.8) dà la misura della mutua informazione media tra le misure di $s(n)$ e quelle di $s(n+T)$. Quando *T* diviene molto grande la caoticità del sistema fa sì che le misure di $s(n)$ e $s(n+T)$ divengano praticamente indipendenti e $I(T)$ tende a zero.

$I(T)$ assume così il ruolo di una funzione di autocorrelazione non lineare ed il criterio per la scelta del *time delay*, suggerito da (Fraser and Swinney, 1986), è di scegliere *T* in corrispondenza del primo minimo della mutua informazione media (2.8). In assenza di minimo per $I(T)$ o quando è noto che i dati provengono da una mappa discreta, Abarbanel, Brown, Sidorowich e Tsimring suggeriscono di porre $T = 1 \text{ o } 2$ (Abarbanel et al., 1993).

Nella Figura 2.1. è rappresentata la funzione di autocorrelazione lineare per la serie dei valori *x* del sistema di Lorenz (in alto) e del sistema di Hénon (in basso). Nel caso del sistema di Lorenz l'autocorrelazione si riduce di un fattore pari a $1/e$ quando $T = 38$ ed il primo zero si ha quando $T \approx 372$. Nel caso del sistema di Hénon si vede come l'autocorrelazione diventi negativa già con *lag* uguale a 1.

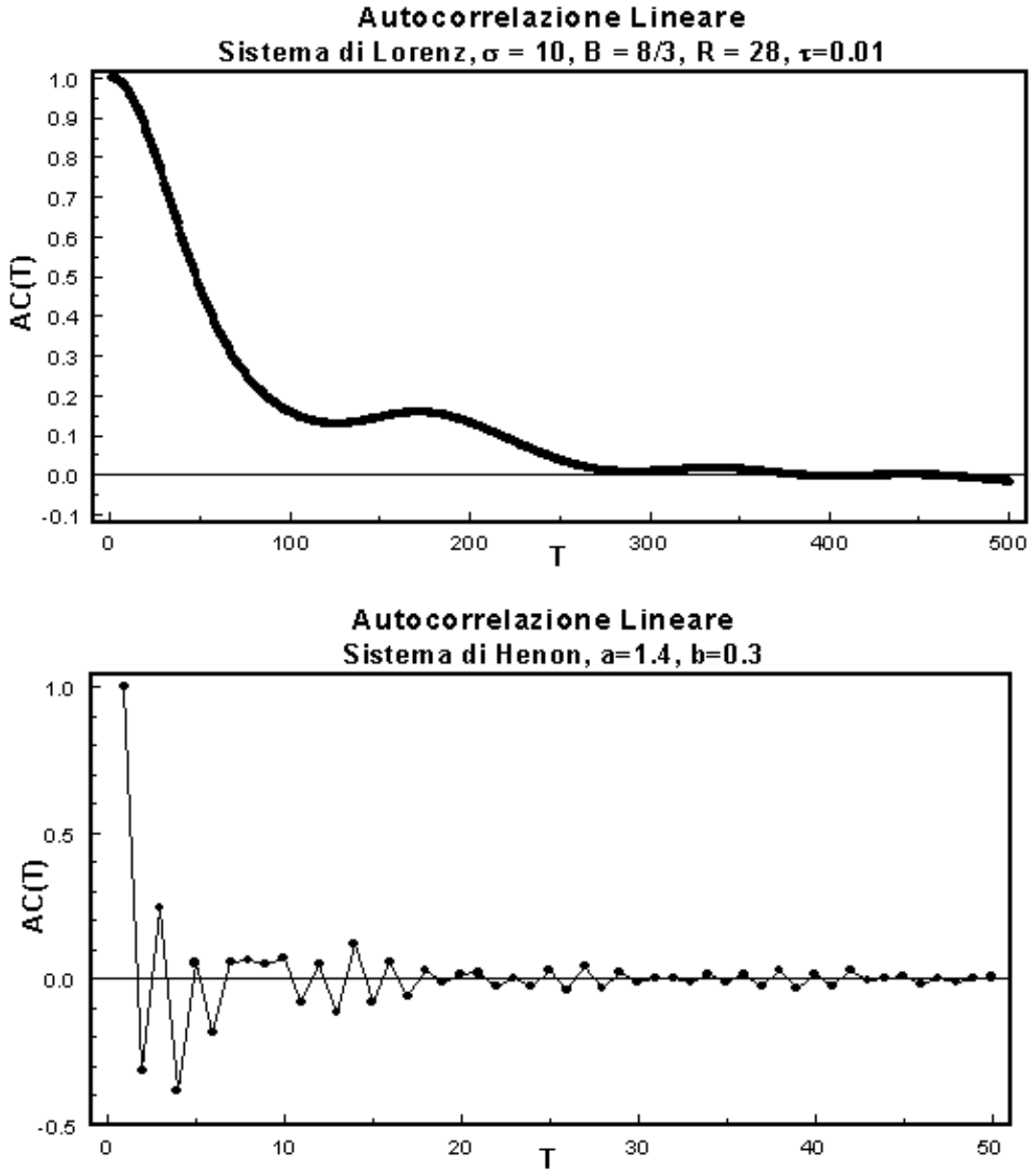

FIGURA 2.1 Autocorrelazione lineare per la serie dei valori x del sistema di Lorenz (in alto) e del sistema di Hénon (in basso).

Nella Figura 2.2 è rappresentata la mutua informazione media calcolata per le stesse serie della Figura 2.1. Per il sistema di Lorenz il primo minimo si ha quando $T = 17$, mentre per il sistema di Hénon la curva non presenta minimo e quindi, in questo caso, la scelta del *time delay* coincide con quella che deriva dall'analisi dell'autocorrelazione: $T = 1$.

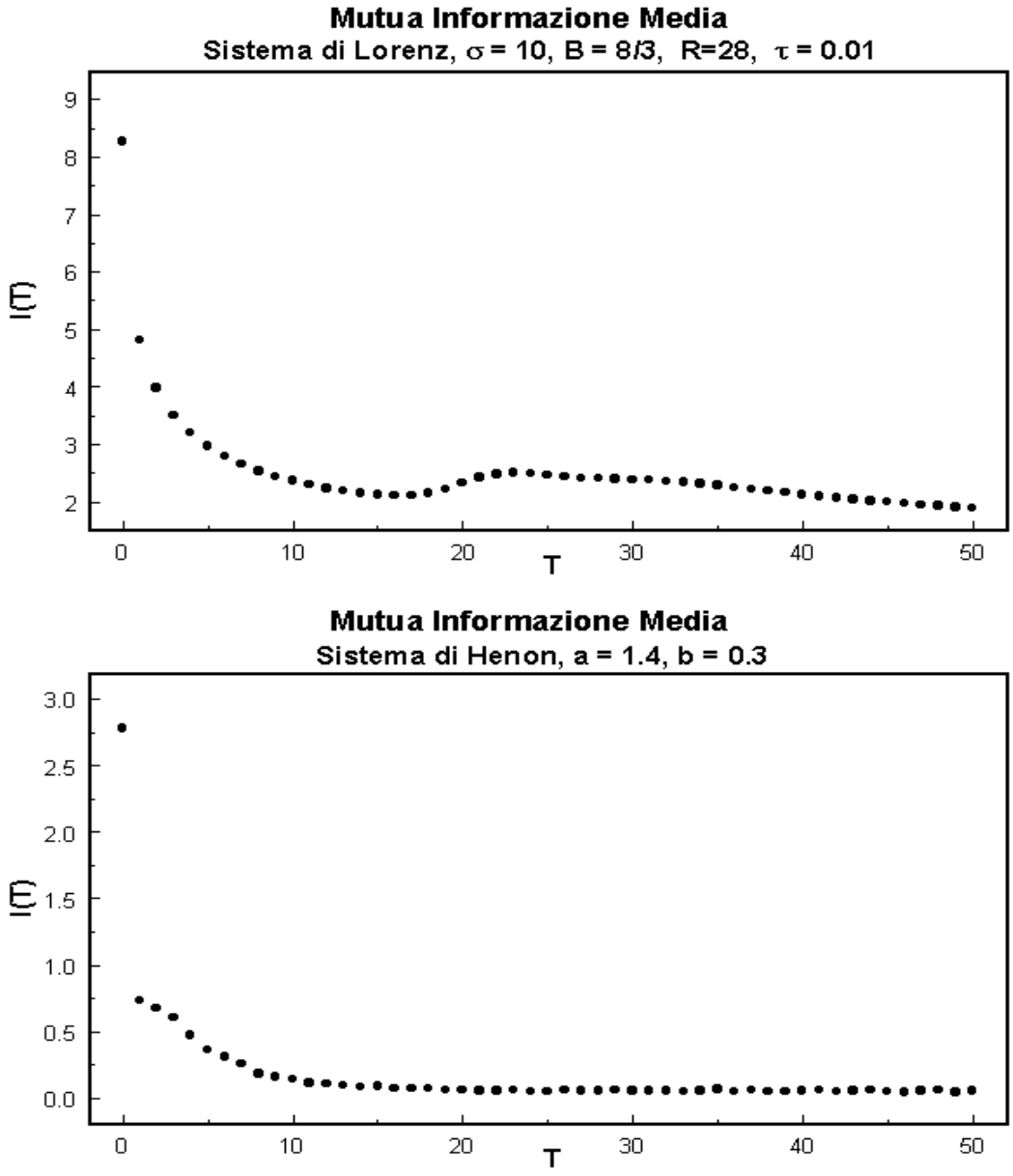

FIGURA 2.2 Mutua informazione media per la serie dei valori x del sistema di Lorenz (in alto) e del sistema di Hénon (in basso).

2.3 Dimensione di Embedding

L'operazione di misurazione corrisponde ad una proiezione sull'asse $s(n)$ di un'orbita del sistema dinamico reale. Ciò avrà come effetto sovrapposizioni dell'orbita che, per il teorema dell'unicità delle soluzioni delle equazioni differenziali, non esistono nello spazio delle variabili dinamiche; nella Figura 2.3, ad

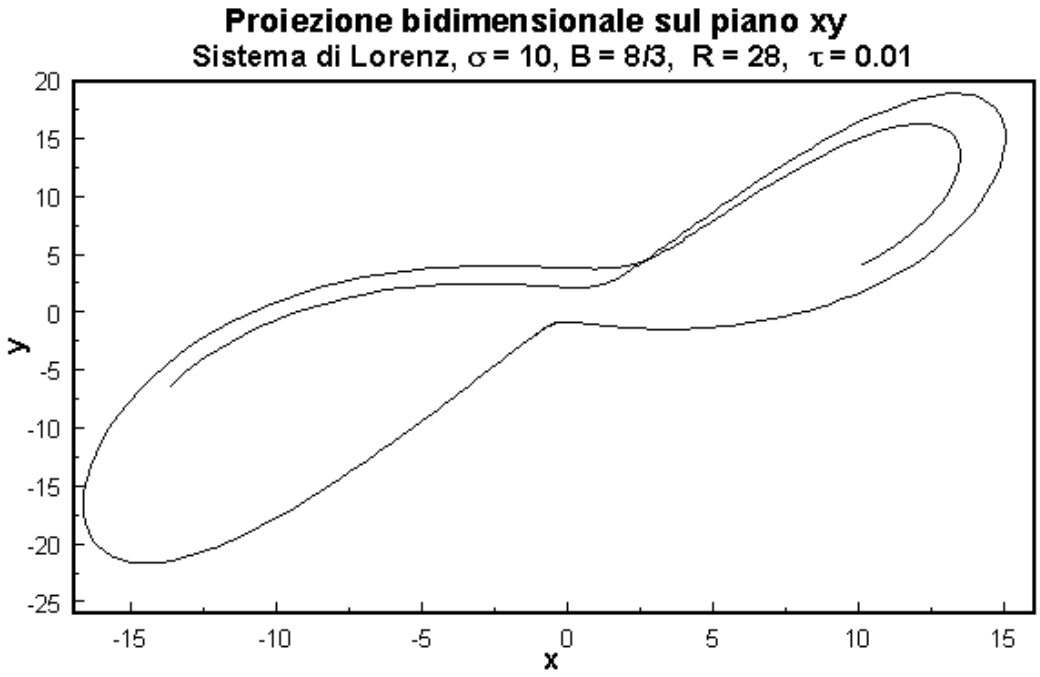

FIGURA 2.3 Proiezione sul piano *xy* di un'orbita del sistema di Lorenz.

esempio, è rappresentata la proiezione sul piano *xy* di un'orbita del sistema di Lorenz: come si vede, a causa della proiezione la curva si interseca in un punto. La ricostruzione dello spazio delle fasi fatta correttamente elimina proprio tali sovrapposizioni fittizie.

Se le orbite sono contenute in uno spazio di dimensione d_A, si può dimostrare che la condizione *sufficiente*, affinché la ricostruzione (2.3) elimini tutte le sovrapposizioni, è che sia $d > 2d_A$. D'altro canto la dimensione *necessaria* a ricostruire efficacemente lo spazio delle fasi è spesso minore di quella sufficiente. Inoltre, in generale, la dimensione d_A non è conosciuta a priori.

Le sovrapposizioni fittizie causate dalla proiezione dei vettori di stato in uno spazio a dimensione *d*, inferiore a quella necessaria per una corretta ricostruzione, fanno sì che un dato punto avrà dei punti vicini che sono tali solo in virtù di tale proiezione ma che potrebbero risultare lontani in uno spazio correttamente ricostruito (falsi vicini).

Per stabilire quando la dimensione di *embedding* scelta è quella necessaria alla corretta ricostruzione si considerano tutti i punti dello spazio ricostruito:

$$\mathbf{y}(k) = [s(k), s(k+T), \ldots, s(k+(d-1)T)]$$

ed i relativi primi vicini:

$$\mathbf{y}^{NN}(k) = [s^{NN}(k), s^{NN}(k+T), \ldots, s^{NN}(k+(d-1)T)],$$

il quadrato della loro distanza euclidea sarà:

$$R_d(k)^2 = \sum_{m=1}^{d} [s(k+(m-1)T) - s^{NN}(k+(m-1)T)]^2 \ . \qquad (2.9)$$

Nello spazio a dimensione $d+1$ tale distanza diviene:

$$\begin{aligned} R_{d+1}(k)^2 &= \sum_{m=1}^{d+1} [s(k+(m-1)T) - s^{NN}(k+(m-1)T]^2 \\ &= R_d(k)^2 + |\, s(k+dT) - s^{NN}(k+dT)\,|^2 \ . \end{aligned} \qquad (2.10)$$

Il rapporto tra la distanza dei punti a dimensione $d+1$ e a quella dei punti a dimensione *d* è dato da:

$$\sqrt{\frac{R_{d+1}(k)^2 - R_d(k)^2}{R_d(k)^2}} = \frac{|\, s(k+dT) - s^{NN}(k+dT)\,|}{R_d(k)} \ . \qquad (2.11)$$

Quando tale rapporto supera una soglia prefissata:

$$\frac{|\, s(k+dT) - s^{NN}(k+dT)\,|}{R_d(k)} > R_T \ . \qquad (2.12)$$

il punto $\mathbf{y}^{NN}(k)$ viene considerato un falso vicino. Per valori di R_T compresi nell'intervallo $10 \le R_T \le 50$, il numero di falsi vicini individuati dalla (2.12) è costante (Abarbanel et al., 1993).

La percentuale di falsi vicini individuati dalla (2.12) diminuirà da un valore iniziale (di solito prossimo al 100%) in corrispondenza di $d = 1$, fino a zero quando la corretta dimensione di *embedding* d_E è stata raggiunta; per $d > d_E$ il numero di falsi vicini resterà zero poiché ormai tutte le false sovrapposizioni saranno state eliminate e l'aumentare le dimensioni non ne genera di nuove.

Un difetto intrinseco al metodo è che, se applicato a dati generati da un sistema stocastico, e quindi con un numero di dimensioni molto elevato, esso indica,

per tali dati, una dimensione di *embedding* bassa. Ciò è dovuto al fatto che i dati generati da un sistema stocastico tendono a popolare uniformemente lo spazio a disposizione. Poiché il volume aumenta in ragione della distanza elevata alla dimensione dello spazio, all'aumentare della dimensione di *embedding* i dati (che sono una quantità finita) si allontaneranno sempre di più l'uno dall'altro e quindi due punti primi vicini in realtà possono essere "lontani" e quindi il denominatore della (2.12) aumenta esponenzialmente fino a che, in corrispondenza ad un dato valore $\overline{d}$, il criterio non è più soddisfatto ed il numero dei falsi vicini scende a zero.

Nella Figura 2.4 è rappresentata la percentuale di falsi vicini per una serie di numeri casuali: è evidente come l'utilizzo del solo criterio (2.12) induce a ritenere, erroneamente, che $\overline{d} = 5$ sia la corretta dimensione di *embedding*.

Per ovviare a questo problema, al criterio (2.12) per determinare un falso vicino occorre aggiungerne un altro: che la distanza aggiunta aumentando la dimensione dello spazio non sia troppo maggiore del "diametro" dell'attrattore. In altri termini se:

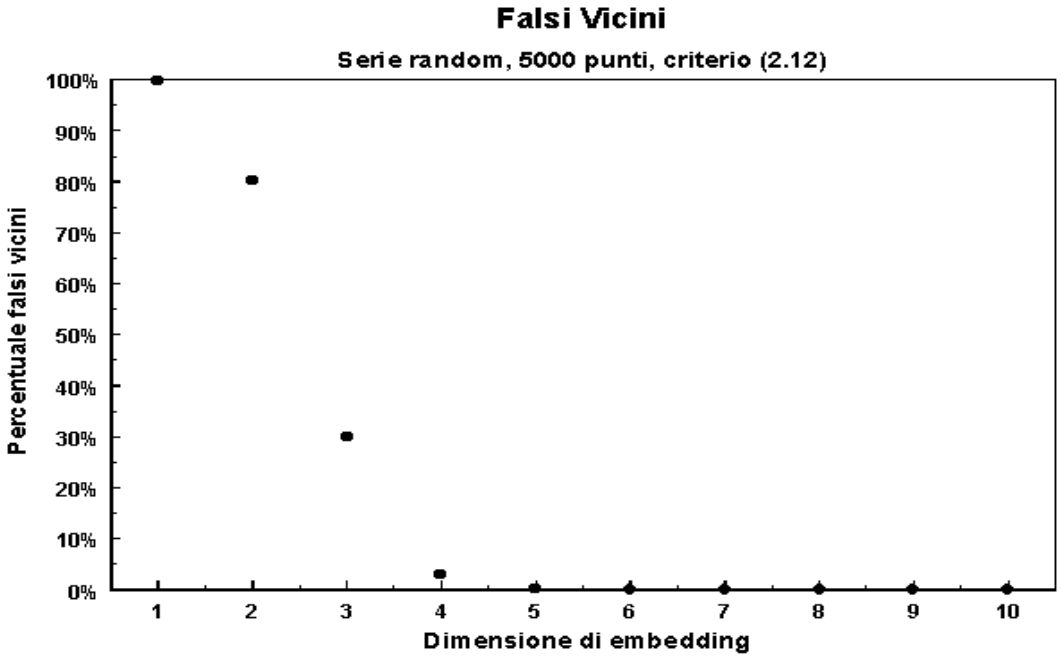

FIGURA 2.4 Percentuale di falsi vicini per una serie di numeri casuali: l'uso del solo criterio (2.12) non è sufficiente a discriminare le serie stocastiche.

$$\frac{|\, s(k+dT) - s^{NN}(k+dT)\,|}{R_A} > 2 \qquad (2.13)$$

ove il "diametro" dell'attrattore viene stimato come lo scarto quadratico medio dei dati attorno alla media:

$$R_A^2 = \frac{1}{N}\sum_{k=1}^{N}[s(k) - \bar{s}]^2 \qquad (2.14)$$

allora il punto $\mathbf{y}^{NN}(k)$ viene considerato un falso vicino.

Nella Figura 2.5, relativa alla stessa serie della Figura 2.4, si vede come l'utilizzo di entrambi i criteri consenta di riconoscere la natura stocastica dei dati.

Nella Figura 2.6 è rappresentata la percentuale di falsi vicini, ottenuta utilizzando entrambi i criteri, per la variabile *x* del sistema di Lorenz (in alto)e per la variabile x del sistema di Hénon (in basso). Il metodo consente di determinare la corretta dimensione di *embedding* in entrambi i casi: $d = 3$ per il sistema di Lorenz e $d = 2$ per quello di Hénon.

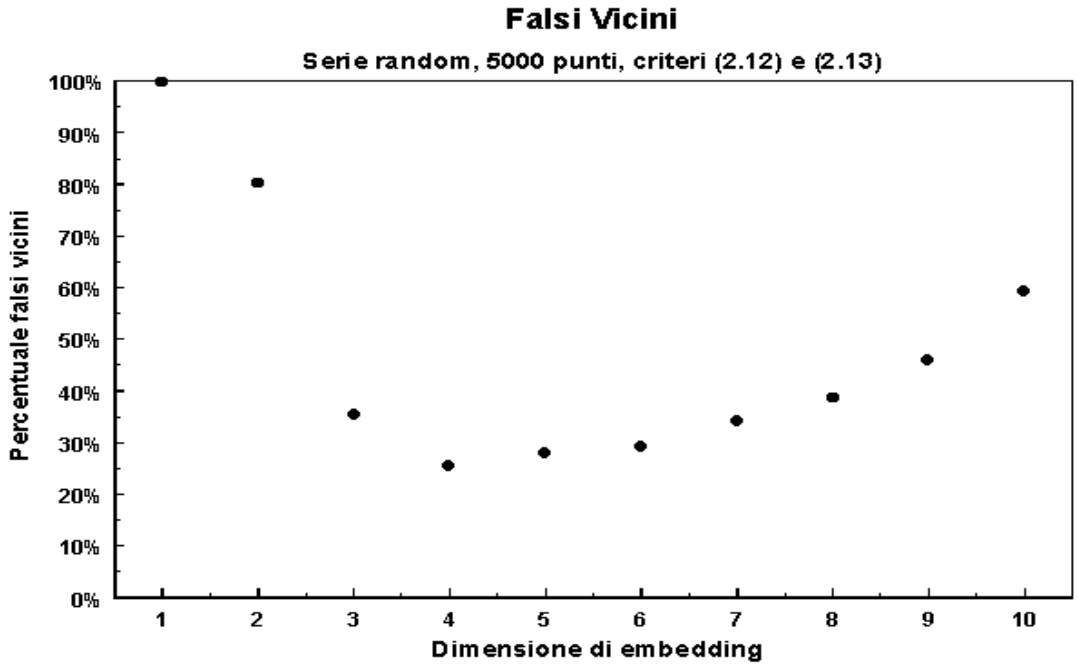

FIGURA 2.5 Percentuale di falsi vicini per la stessa serie di numeri casuali di Figura 2.4: l'uso di entrambi i criteri consente di discriminare la serie stocastica.

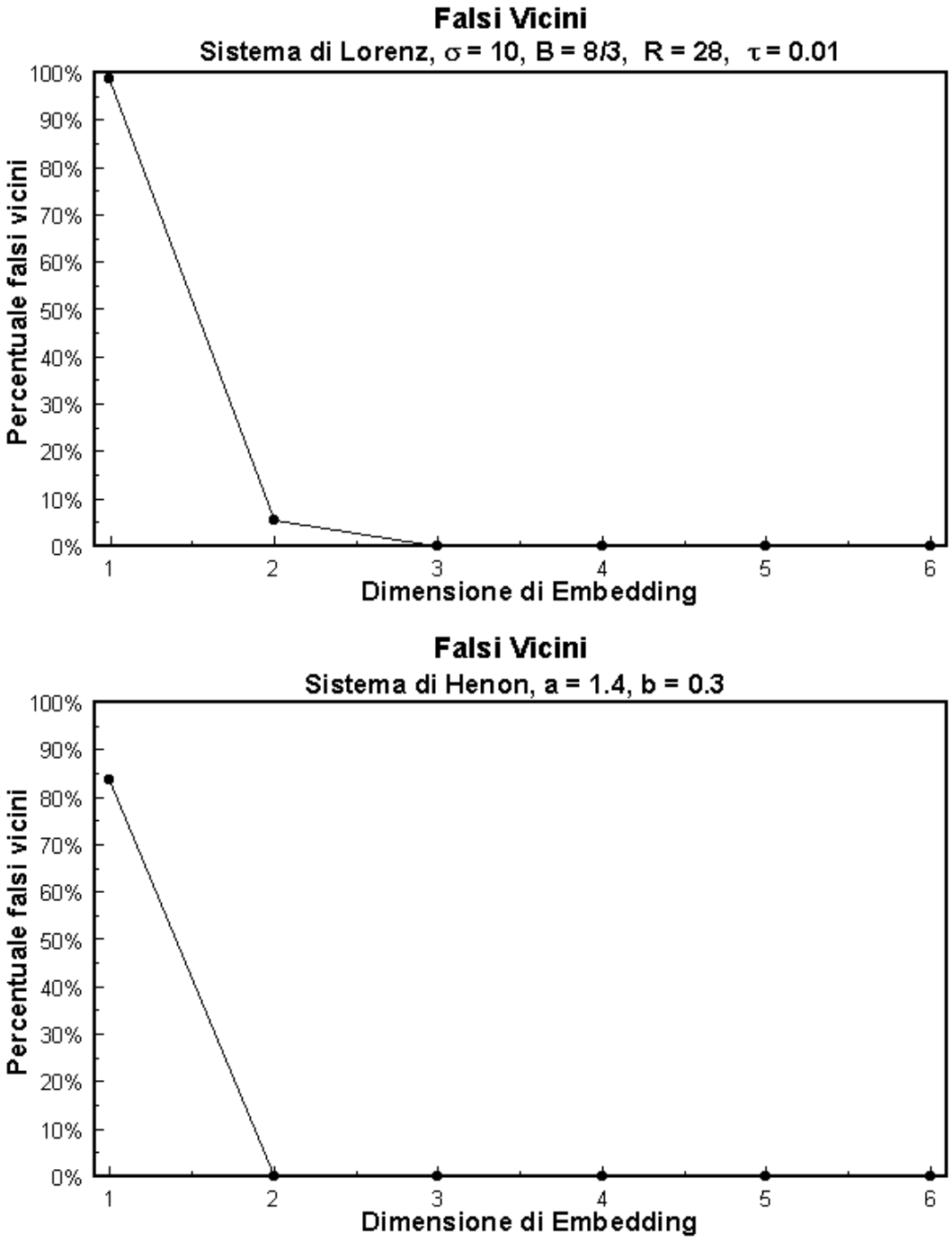

FIGURA 2.6 Percentuale di falsi vicini per il sistema di Lorenz (in alto) e per quello di Hénon (in basso).

Con il *time delay*, che utilizzando il metodo della mutua informazione media è risultato essere pari a 0.17 per il sistema di Lorenz e pari a 1 per il sistema di Hénon; e la dimensione di *embedding*, che utilizzando il metodo dei falsi vicini è risultata essere pari a 3 e pari a 2 rispettivamente, è possibile ricostruire gli attrattori

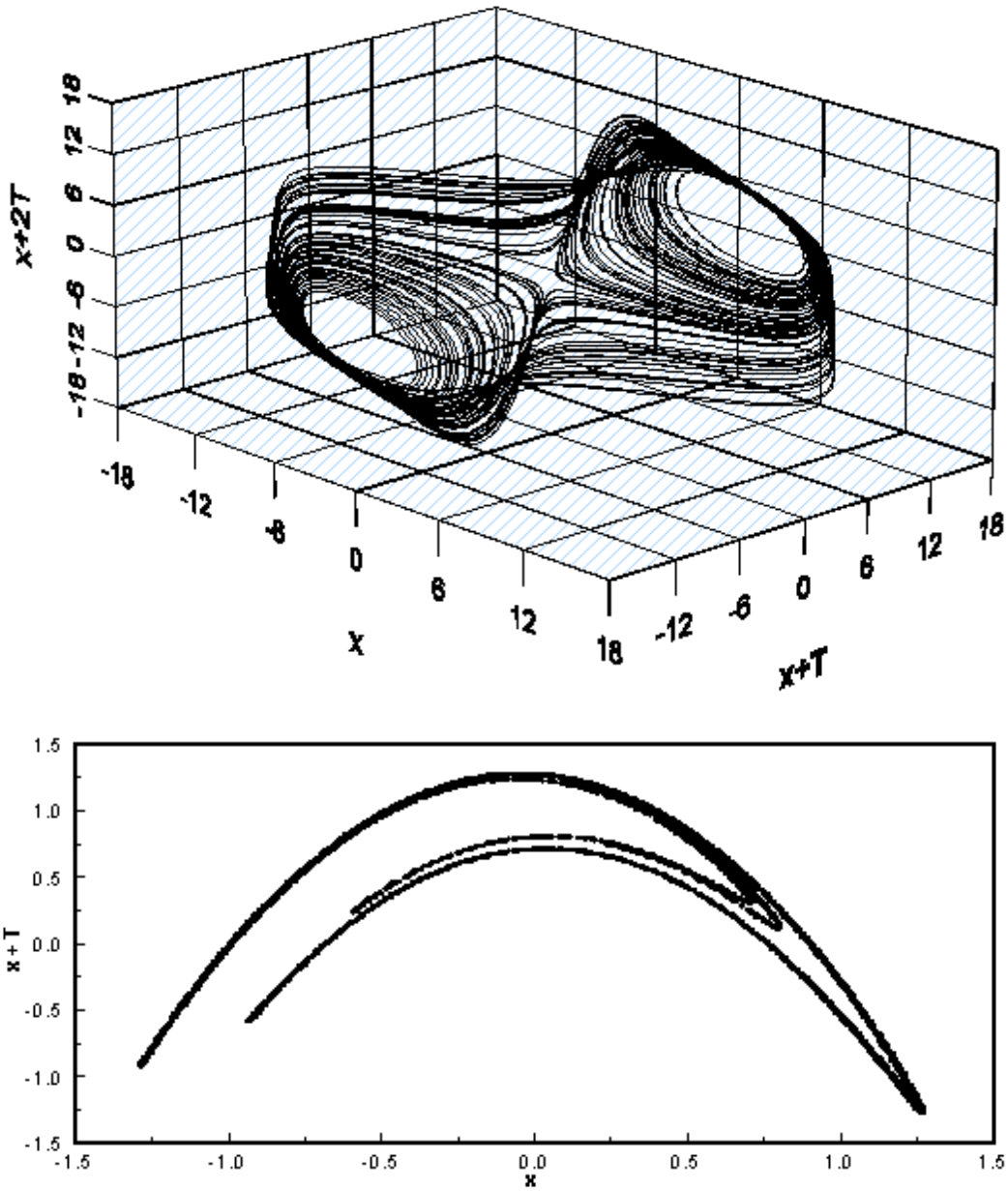

FIGURA 2.7 Attrattore ricostruito del sistema di Lorenz. (in alto) e del sistema di Hénon (in basso)

dei due sistemi sulla base dei soli dati della variabile *x*. Ovviamente la forma dell'attrattore ricostruito è diversa da quella dell'attrattore del sistema originario ma, in virtù del teorema di *embedding*, i due attrattori hanno le stesse proprietà dinamiche.

Nella Figura 2.7 sono rappresentati gli attrattori ricostruiti del sistema di Lorenz (in alto) e di quello di Hénon (in basso).

2.3 Dimensione di Embedding

Gli attrattori sono formati dalle orbite dei vettori si stato $\mathbf{y}$ dati da:

$$\mathbf{y}(n) = [x(n), x(n+T), x(n+2T)] \qquad n = 1,2,\ldots$$

dove $x(n)$ è la prima variabile del sistema.

3. Invarianti della dinamica

3.1 Dimensione frattale

La dimensione frattale è una caratteristica della geometria dell'attrattore ed è in relazione con il modo in cui i punti sono distribuiti nello spazio di *embedding* a *d* dimensioni.

Tra la fine dell'ottocento e l'inizio del novecento uno dei principali problemi della matematica era di determinare che cosa esattamente fosse e che proprietà avesse la dimensione di uno spazio; di essa vennero formulate diverse definizioni: dimensione topologica, dimensione di Hausdorff, dimensione frattale, dimensione di autosomiglianza, *box-counting dimension*, dimensione di capacità, dimensione di informazione, dimensione euclidea e altre.

Tutte queste definizioni sono in qualche modo legate tra loro; alcune hanno senso solo in specifiche situazioni e non in altre; in certi casi le diverse definizioni coincidono, in altri non tutte conducono allo stesso risultato numerico.

All'origine del concetto di dimensione frattale c'è la definizione di Hausdorff che risale al 1918.

3.1.1 Dimensione di Hausdorff

Dato un sottoinsieme A di $\mathbf{R}^n$, una famiglia di sottoinsiemi aperti di $\mathbf{R}^n$ $\{U_1, U_2, U_3, \ldots\}$ è detta un *ricoprimento aperto* di A se

$$A \subset \bigcup_{i=1}^{\infty} U_i \, .$$

Dati i numeri reali positivi s ed ε, si definisce:

$$h_\varepsilon^s(A) = \inf\left\{ \sum_{i=0}^{\infty} \operatorname{diam}(U_i)^s \;\middle|\; \begin{array}{l} \{U_1, U_2, \ldots\} \text{ sia un ricoprimento} \\ \text{aperto di } A \text{ con } \operatorname{diam}(U_i) < \varepsilon \end{array} \right\} \qquad (3.1)$$

ove

$$\operatorname{diam}(U) = \sup\{ d(\mathbf{x}_1, \mathbf{x}_2) \mid \mathbf{x}_1, \mathbf{x}_2 \in U \},$$

è il *diametro* dell'insieme U e $d(\mathbf{x}_1, \mathbf{x}_2)$ è la distanza euclidea in $\mathbf{R}^n$.

Nella (3.1) la somma può essere finita o infinita e l'estremo inferiore è esteso a tutti i ricoprimenti aperti di A per i quali gli insiemi U_i hanno diametro minore di ε. Al diminuire di ε diminuiscono i possibili ricoprimenti di A e quindi l'estremo inferiore aumenta tendendo ad un limite, per $\varepsilon \to 0$, che può essere infinito o un numero reale.

Tale limite:

$$h^s(A) = \lim_{\varepsilon \to 0} h_\varepsilon^s(A) \qquad (3.2)$$

è detto la *misura s-dimensionale di Hausdorff dell'insieme A*.

Hausdorff dimostrò che, per ogni insieme A, esiste un numero $D_H(A)$ tale che:

$$h^s(A) = \begin{cases} \infty & \text{per } s < D_H(A) \\ 0 & \text{per } s > D_H(A) \end{cases} \qquad (3.3)$$

Il numero $D_H(A)$, estremo inferiore dei valori di *s* per cui $h^s(A) = \infty$ ed estremo superiore dei valori di *s* per cui $h^s(A) = 0$, è detto *dimensione di Hausdorff dell'insieme A*:

$$D_H(A) = \inf\{s \mid h^s(A) = 0\} = \sup\{s \mid h^s(A) = \infty\}. \qquad (3.4)$$

Se $s = D_H(A)$ allora $h^s(A)$ può essere zero, infinito o un numero reale positivo.

Alcune fondamentali proprietà della dimensione di Hausdorff sono le seguenti:

i. Se $A \subset R^n$ allora $D_H(A) \le n$.

ii. Se $A \subset B$ allora $D_H(A) \le D_H(B)$.

iii. Se *A* è un insieme numerabile allora $D_H(A) = 0$.

iv. Se $A \subset R^n$ e $D_H(A) < 1$ allora *A* è totalmente disconnesso.

v. Sia C_∞ l'insieme di Cantor[4], allora $D_H(C_\infty) = \log 2 / \log 3$.

La dimensione di Hausdorff, benché storicamente molto importante, è di scarso interesse pratico poiché è molto difficile da calcolare in casi elementari e praticamente impossibile da stimare in casi concreti. La difficoltà principale è data dal calcolo dei termini $\sum_{i=0}^{\infty} \mathrm{diam}(U_i)^s$. La *box-counting dimension* semplifica il problema rimpiazzando tali termini.

3.1.2 Box-counting dimension

Dato un sottoinsieme limitato *A* di $\mathbf{R}^n$ sia $N_\delta(A)$ il più piccolo numero di insiemi di diametro minore o uguale a δ che ricopre *A*[5]. Allora la *box-counting dimension* è definita come:

$$D_b(A) = \lim_{\delta \to 0} \frac{\log N_\delta(A)}{\log 1/\delta}. \qquad (3.5)$$

---

[4] L'insieme di Cantor è un insieme infinito di punti nell'intervallo unitario [0,1] che si ottiene dividendo tale intervallo in tre parti uguali, togliendo la parte centrale ed iterando all'infinito la procedura alle parti rimanenti.

[5] Poiché *A* è limitato, si può assumere che tale numero sia finito.

3.1 Dimensione frattale

Esistono diverse definizioni equivalenti di $D_b(A)$. Per esempio, suddividendo $\mathbf{R}^n$ mediante un reticolo di passo δ e indicato con $N'_\delta(A)$ il numero di "cubetti" che interseca A, si può verificare che:

$$D_b(A) = \lim_{\delta \to 0} \frac{\log N'_\delta(A)}{\log 1/\delta}. \qquad (3.6)$$

In sostanza la (3.5) afferma che $N_\delta(A) \propto \delta^{-s}$ per piccoli δ, dove $s = D_b(A)$. D'altro canto:

$$N_\delta(A)\delta^s = \inf\left\{ \sum_i \delta^s \;\middle|\; \begin{matrix} \{U_1, U_2, \ldots\} \text{ sia un ricoprimento} \\ \text{aperto di } A \text{ con } \mathrm{diam}(U_i) < \delta \end{matrix} \right\} \qquad (3.7)$$

e quindi, confrontando la (3.7) e la (3.1) si vede come la differenza stia proprio nei termini $\sum_{i=0}^{\infty} \mathrm{diam}(U_i)^s$ sostituiti dai termini $\sum_i \delta^s$.

L'algoritmo per il calcolo della *box-counting dimension* è apparentemente semplice; nel caso dell'attrattore di Lorenz si potrebbe procedere come segue:

1. si determina un parallelepipedo che contenga l'attrattore,
2. si suddivide tale regione in cubetti di lato δ,
3. si conta il numero di cubetti, $N(\delta)$, che contiene almeno un punto dell'attrattore,
4. si ripete la procedura per valori decrescenti di δ,

il numero di cubetti contenenti punti dell'attrattore è proporzionale ad una qualche potenza di δ:

$$N(\delta) \propto \delta^{-D_b} \qquad (3.8)$$

ove D_b è la *box-counting dimension*. Quindi, rappresentando $\log N(\delta)$ in funzione di $\log 1/\delta$, si ottiene una retta la cui pendenza fornisce la grandezza cercata.

Purtroppo il fatto che nel caso di dati sperimentali il numero di punti sull'attrattore sia finito, implica che esiste un particolare valore δ^* tale che, se $\delta < \delta^*$, $N(\delta)$ assume un valore costante pari al numero di punti sull'attrattore[6].

[6] Basta porre δ^* pari alla minima distanza tra due punti qualunque dell'attrattore.

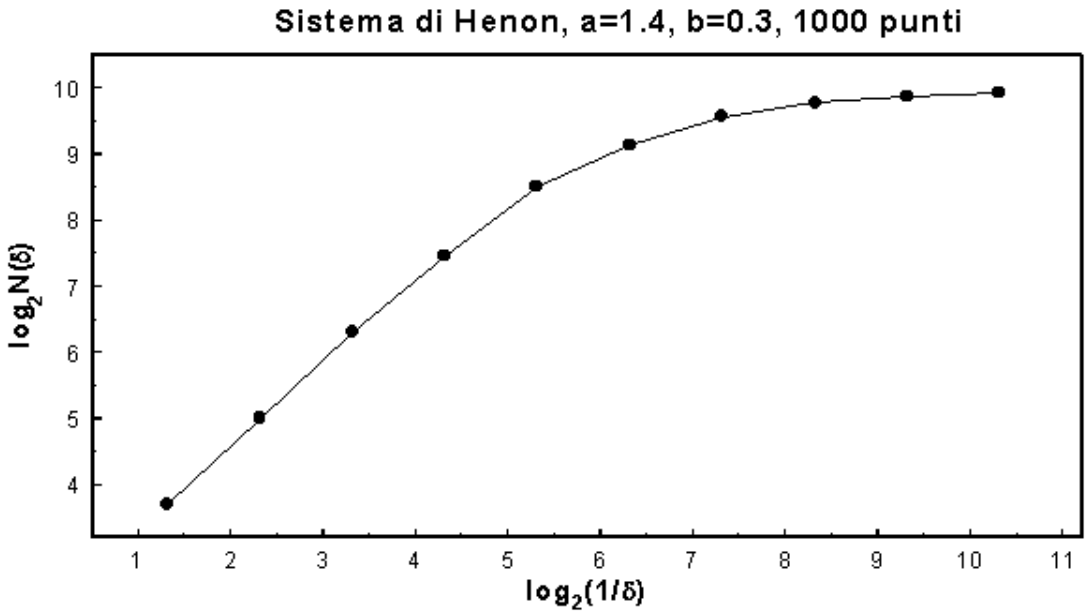

FIGURA 3.1 $\log_2 N(\delta)$ in funzione di $\log_2 1/\delta$ per il sistema di Hénon. Sono stati utilizzati solo 1000 punti per rendere più evidente la saturazione della curva.

Dunque, benché più semplice da calcolare della dimensione di Hausdorff, anche la stima della *box-counting dimension* risulta piuttosto problematica.

Nella Figura 3.1 è rappresentato il grafico di $\log_2 N(\delta)$ in funzione di $\log_2 1/\delta$ per il sistema di Hénon: appare evidente come, al diminuire di δ, la curva saturi rendendo così impossibile la determinazione della pendenza nella zona ad alta risoluzione. D'altro canto un *fitting* della sola zona lineare a bassa risoluzione indurrebbe un errore sistematico nella stima della dimensione che, in base alla definizione (3.5), deve essere valutata per valori di δ piccoli.

Un ulteriore limite della *box-counting dimension* è che un cubetto che interseca l'attrattore viene contato una sola volta indipendentemente dal numero di punti contenuti e dunque tale grandezza non riflette in alcun modo la distribuzione dei punti all'interno dell'orbita.

La *dimensione di informazione* tiene conto del peso di ciascun cubetto e considera il grado di disomogeneità nella distribuzione dei punti rispetto allo spazio ricoperto dall'attrattore.

3.1.3 Dimensione di informazione

Dato un sottoinsieme aperto *B* dello spazio che contiene l'attrattore del sistema, la percentuale dei punti dell'attrattore contenuti in *B* è detta *misura naturale* per il sistema:

$$\mu(B) = \lim_{n\to\infty} \frac{1}{n+1} \sum_{k=0}^{n} U_B(x_k) \tag{3.9}$$

dove:

$$U_B(x) = \begin{cases} 1 \text{ se } x \in B \\ 0 \text{ altrimenti} \end{cases}.$$

La misura naturale quantifica la "massa" di una data porzione dell'attrattore.

Volendo dunque tenere conto del peso di ciascun cubetto la grandezza $\log N(\delta)$ va sostituita dalla:

$$I(\delta) = \sum_{k=1}^{N(\delta)} \mu(B_k) \log_2 \frac{1}{\mu(B_k)} \tag{3.10}$$

ove la somma è estesa ai soli $N(\delta)$ insiemi (o "cubetti" di lato δ) che intersecano l'attrattore.

La grandezza $I(\delta)$ rappresenta la quantità di informazione necessaria a determinare la posizione di un punto nell'attrattore con un'accuratezza pari a δ[7].

Al diminuire di δ $I(\delta)$ aumenta e se si rappresenta $I(\delta)$ in funzione di $\log_2 1/\delta$, si vede che le due grandezze sono legate da una legge di potenza:

$$I(\delta) \approx I_0 + D_i \log_2 \frac{1}{\delta}. \tag{3.11}$$

[7] Volgarizzando la Teoria dell'Informazione di Shannon, si può dire che $I(\delta)$ rappresenta il numero medio di domande, le cui risposte possono essere solo si o no, necessario a determinare se un dato punto appartiene ad un dato insieme. Il logaritmo in base 2 nella (3.11) offre il vantaggio che $I(\delta)$ risulta così espressa in unità di bits.

ove I_0 è una costante e D_i, che quantifica l'aumento di informazione ottenuto raddoppiando la risoluzione del reticolo che ricopre l'attrattore, è, per definizione, la *dimensione di informazione*.

Nella Figura 3.2 è rappresentato il grafico di $I(\delta)$ in funzione di $\log_2 1/\delta$ per il sistema di Hénon. Il problema della saturazione, che la dimensione di informazione riduce ma comunque non elimina, è meno evidente anche perché sono stati utilizzati 10000 punti rispetto ai 1000 della Figura 3.1.

Anche in questo caso l'algoritmo, pur essendo concettualmente semplice, è di difficile applicazione ai dati sperimentali. La procedura infatti, già nel caso bidimensionale dell'attrattore di Hénon, richiede un uso intensivo e prolungato del calcolatore (dell'ordine dell'ora per 10000 punti e 7 valori di δ fino ad un minimo pari a circa 2^{-8}, grandezze comunque insufficienti) e, soprattutto, un numero molto elevato di punti (dell'ordine di 10^7 per una stima attendibile) tale da renderla inapplicabile a sistemi dinamici, soprattutto ambientali, che possono avere quattro o più gradi di libertà.

Il valore della dimensione di informazione che si desume dai dati di Figura

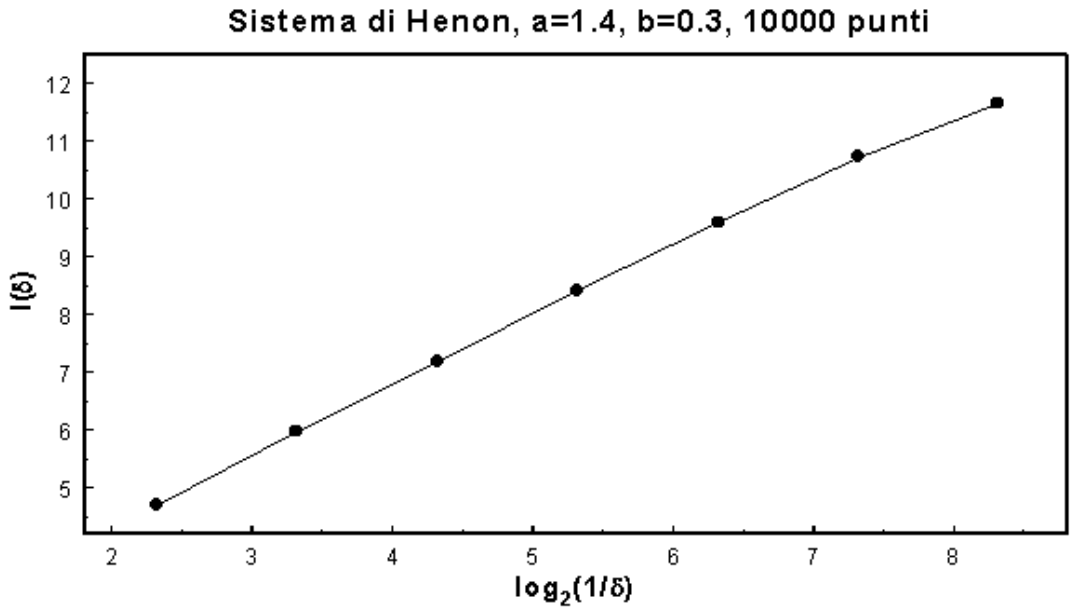

FIGURA 3.2 $I(\delta)$ in funzione di $\log_2 1/\delta$ per il sistema di Hénon.

3.2 è pari a

$$D_i = 1.17 \pm 0.02 .$$

Tale valore è sottostimato a causa del numero di punti non sufficiente infatti Peitgen, Jürgens e Supe (1992) usando 10^8 punti ottengono:

$$D_i = 1.23 \pm 0.02 .$$

Il fatto che Grassberger (1983), in seguito a calcoli piuttosto elaborati e sofisticati, abbia ottenuto il valore

$$D_b = 1.28 \pm 0.01$$

per la *box-counting dimension* dell'attrattore di Hénon, non è in contraddizione con il risultato di Peitgen; tra la dimensione di informazione e la *box-counting dimension* sussiste infatti la seguente relazione:

$$D_i \leq D_b . \qquad (3.12)$$

La (3.12) deriva dalla fondamentale disuguaglianza tra la media geometrica e quella aritmetica di un insieme di numeri $a_1, a_2, \ldots, a_N$ positivi:

$$\sqrt[N]{\prod_{k=1}^{N} a_k} \leq \frac{1}{N} \sum_{k=1}^{N} a_k . \qquad (3.13)$$

Se con p_k si indicano N numeri positivi tali che $p_1 + \ldots + p_N = 1$ allora, in base alla (3.13), si ha:

$$\prod_{k=1}^{N} a_k^{p_k} \leq \sum_{k=1}^{N} p_k a_k . \qquad (3.14)$$

dove vale l'uguaglianza solo se gli a_k sono tutti uguali.

Posto $a_k = 1/p_k$ e $N = N(\delta)$, la (3.14) diviene:

$$\prod_{k=1}^{N(\delta)} \left(\frac{1}{p_k} \right)^{p_k} \leq \sum_{k=1}^{N(\delta)} p_k \frac{1}{p_k} = N(\delta) .$$

Se le p_k indicano la misura naturale $\mu(B_k)$, passando ai logaritmi si ha:

$$I(\delta) = \sum_{k=1}^{N(\delta)} p_k \log_2 \frac{1}{p_k} \leq \log_2 N(\delta) .$$

Da cui, in base alle definizioni

$$D_i = \lim_{\delta \to 0} \frac{I(\delta)}{\log_2 1/\delta} \quad \text{e} \quad D_b = \lim_{\delta \to 0} \frac{\log_2 N(\delta)}{\log_2 1/\delta} ,$$

si ricava la (3.12).

3.1.4 Dimensione generalizzata e dimensione di correlazione

Sia la *box-counting dimension* che la dimensione di informazione sono contenute in un insieme di definizioni più generale noto come *dimensioni di Rényi* (1970). Queste si ottengono sostituendo l'informazione $I(\delta)$ della (3.10) con l'informazione del q-esimo ordine di Rényi:

$$I_q(\delta) = \frac{1}{1-q} \log_2 \sum_{k=1}^{N(\delta)} p_k^q \qquad (3.15)$$

ove $q \geq 0$, $p_k = \mu(B_k)$ e la somma è estesa ai soli $N(\delta)$ insiemi che intersecano l'attrattore.

La dimensione di Rényi D_q è definita come:

$$D_q = \lim_{\delta \to 0} \frac{I_q(\delta)}{\log_2 1/\delta} . \qquad (3.16)$$

Se $q = 0$ dalla (3.16) si ricava:

$$I_0(\delta) = \log_2 N(\delta)$$

e dunque la dimensione di Rényi D_0 coincide con la *box-counting dimension*:

$$D_0 = D_b . \tag{3.17}$$

Se $q = 1$ l'informazione (3.16) si può calcolare con la regola di l'Hospital:

$$I_1(\delta) = -\lim_{q \to 1} \frac{\sum p_k^q \ln p_k}{\sum p_k^q \ln 2},$$

e ricordando che $\sum p_k = 1$, si ottiene infine:

$$I_1(\delta) = \sum_{k=1}^{N(\delta)} p_k \log_2 \frac{1}{p_k} . \tag{3.18}$$

La (3.18) è proprio l'informazione (3.10) e quindi si ha che la dimensione di Rényi D_1 coincide con la dimensione di informazione:

$$D_1 = D_i . \tag{3.19}$$

In accordo con la (3.12), la dimensione di Rényi è una funzione decrescente di *q*, cioè:

$$D_p \geq D_q \quad \text{se} \quad p < q .$$

Se D_q diminuisce strettamente all'aumentare di *q*, allora l'attrattore è detto *multifrattale.*

In generale, il termine p_k^q nella (3.15) rappresenta la probabilità che *q* punti scelti a caso nell'attrattore appartengano al medesimo insieme B_k e la somma $\sum p_k^q$ è la probabilità che i *q* punti appartengano tutti ad uno qualunque degli insiemi B_k. Questa probabilità è proporzionale al numero relativo di insiemi di *q* punti dell'attrattore, $C_q(\delta)$, le cui distanze reciproche siano tutte inferiori a δ. In altri termini, dati *m* punti sull'attrattore $x_0, x_1, \ldots, x_m$, sia $T_q(m, \delta)$ il numero di insiemi di *q* punti $(x_{i_1}, x_{i_2}, \ldots, x_{i_q})$ che soddisfano alla:

$$d(x_{i_k}, x_{i_l}) < \delta \quad \text{per} \quad k, l = 1, \ldots, q$$

dove $d(x, y)$ è la distanza euclidea. Allora:

$$C_q(\delta) = \lim_{m\to\infty} \frac{T_q(m,\delta)}{m^q},$$

e si ha:

$$\sum_{k=1}^{N(\delta)} p_k^q \propto C_q(\delta).$$

In base a (3.15) e (3.16), si ricava infine:

$$D_q = \frac{1}{1-q} \lim_{\delta\to 0} \frac{\log_2 C_q(\delta)}{\log_2 1/\delta}. \qquad (3.20)$$

Quando $q = 2$, $C_2(\delta)$ rappresenta una sorta di correlazione tra coppie di punti dell'attrattore ed è detta *funzione di correlazione*, analogamente la dimensione D_2 è nota come *dimensione di correlazione*.

3.2 L'algoritmo di Grassberger-Procaccia

Nel 1983 Grassberger e Procaccia (1983a, 1983b) suggerirono di valutare la dimensione di correlazione per stabilire sia la dimensione di *embedding* che la eventuale dimensione frattale di un sistema dinamico, inaugurando così quel campo della ricerca volto a determinare la caoticità dei più diversi sistemi: dalla meccanica dei fluidi alla fisica dello stato solido, alla meteorologia, all'economia.

L'algoritmo proposto per il calcolo della dimensione di correlazione è il seguente:

1. Fissato il *time delay T* (par. 2.2), per valori crescenti, a partire da 1, della dimensione di *embedding* si calcola la grandezza $C_2(r)$ in base alla:

$$C_2(r) = \frac{2}{N(N-1)} \sum_{i\neq j} \theta(r - |\mathbf{y}(j) - \mathbf{y}(i)|) \qquad (3.21)$$

(mantenendo la notazione più diffusa sostituiamo δ con r) ove le $\mathbf{y}(i)$, $i = 1,\ldots,N$, sono i punti dello spazio delle fasi ricostruito e θ è la funzione di Heaviside:

$$\theta(x) = \begin{cases} 1 \text{ se } x > 0 \\ 0 \text{ se } x < 0 \end{cases}.$$

2. Il diagramma di $\log C_2(r)$ in funzione di $\log r$ risulta formato, al variare della dimensione di *embedding m*, da un fascio di curve con un ampio tratto rettilineo; la pendenza di tale regione lineare è una funzione crescente di *m* sino a che, se il sistema è caotico, viene raggiunta la corretta dimensione di embedding[8]; a questo punto la pendenza satura ad un valore che rappresenta la dimensione di correlazione. Se il sistema non è caotico la pendenza dei tratti rettilinei cresce indefinitamente.

Il metodo illustrato consente di discriminare sistemi stocastici da sistemi caotici e, se esiste l'attrattore, permette di determinarne sia la dimensione di *embedding* che la dimensione frattale (di correlazione). Tale metodo, tuttavia, presenta problemi che ne limitano l'applicabilità, come vedremo in seguito.

Nelle Figure 3.4 e 3.5 sono rappresentate le funzioni di correlazione, per diversi valori della dimensione di embedding *m*, relative alla variabile *x* del sistema di Lorenz e alla variabile *x* del sistema di Hénon.

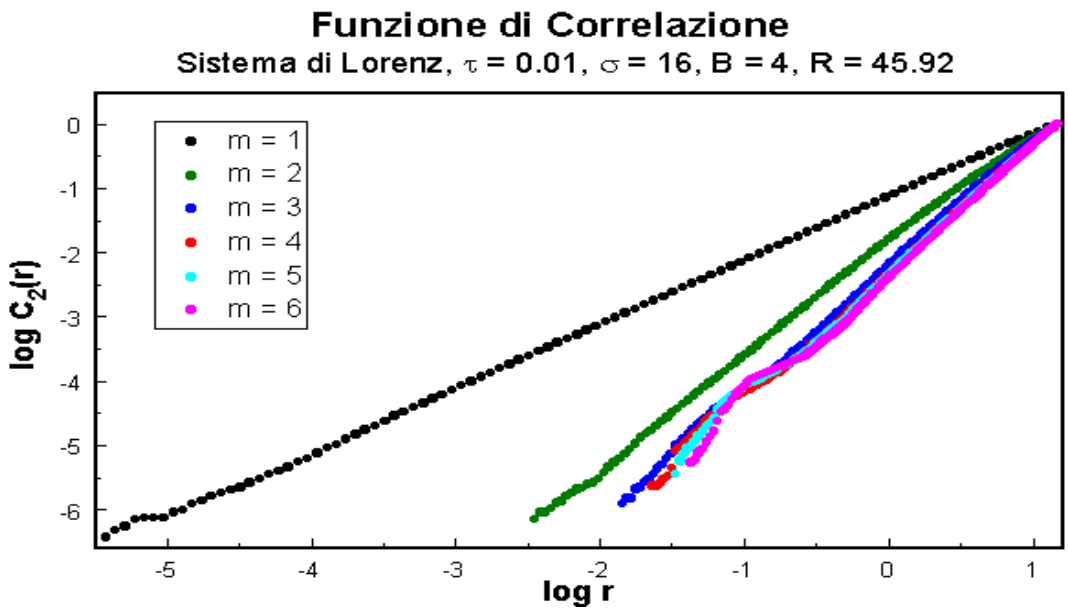

FIGURA 3.4 Funzione di correlazione per la variabile x del sistema di Lorenz relativa a sei valori della dimensione di embedding. Sono stati utilizzati 10000 punti.

[8] Cfr. par. 2.3.

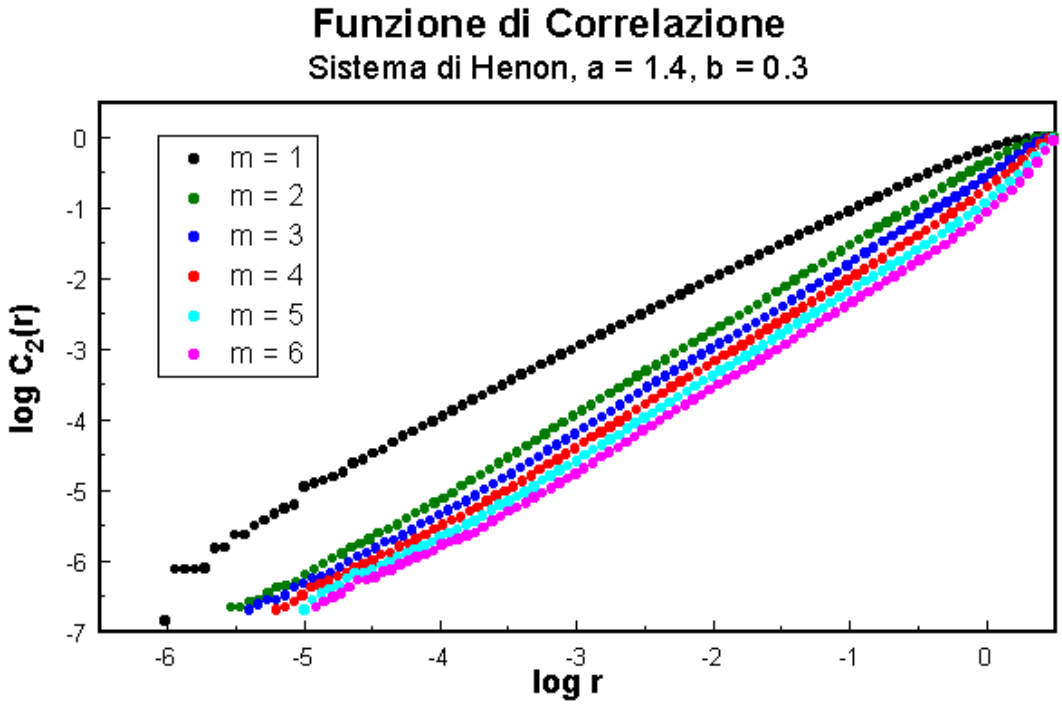

FIGURA 3.5 Funzione di correlazione per la variabile x del sistema di Hénon relativa a sei valori della dimensione di embedding. Sono stati utilizzati 10000 punti.

Come si vede dalle figure, anche nel caso ideale di dati generati al computer nella quantità desiderata e privi di rumore, per poter determinare le pendenze occorre stabilire l'estensione della zona lineare di ciascuna curva, che diventa sempre più

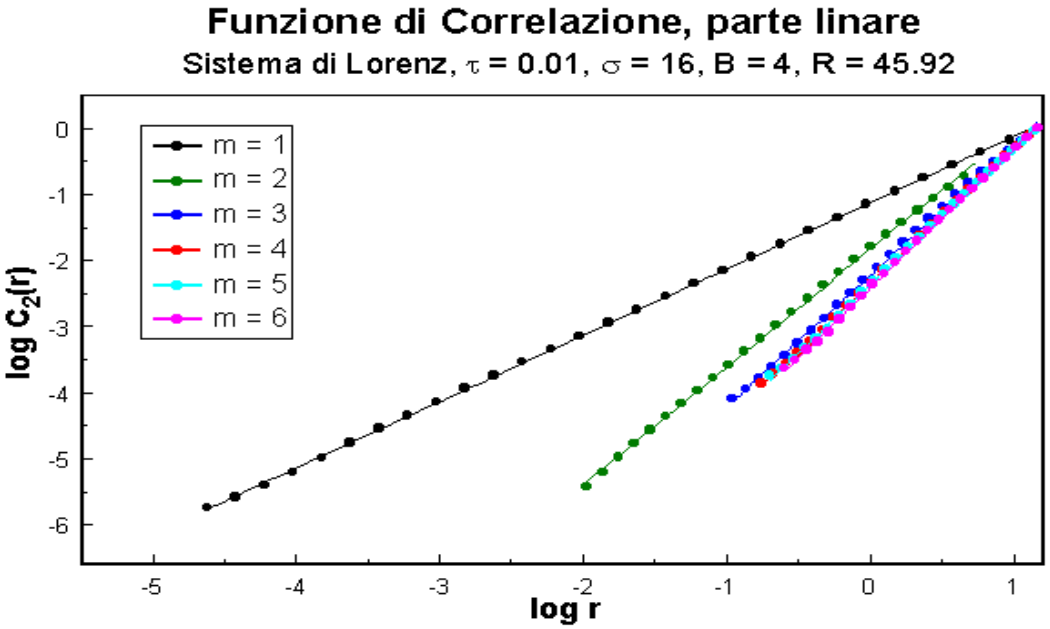

FIGURA 3.6 Zone lineari delle curve di Figura 3.4. Per rendere visibili le rette di interpolazione è stato rappresentato solo un punto ogni tre.

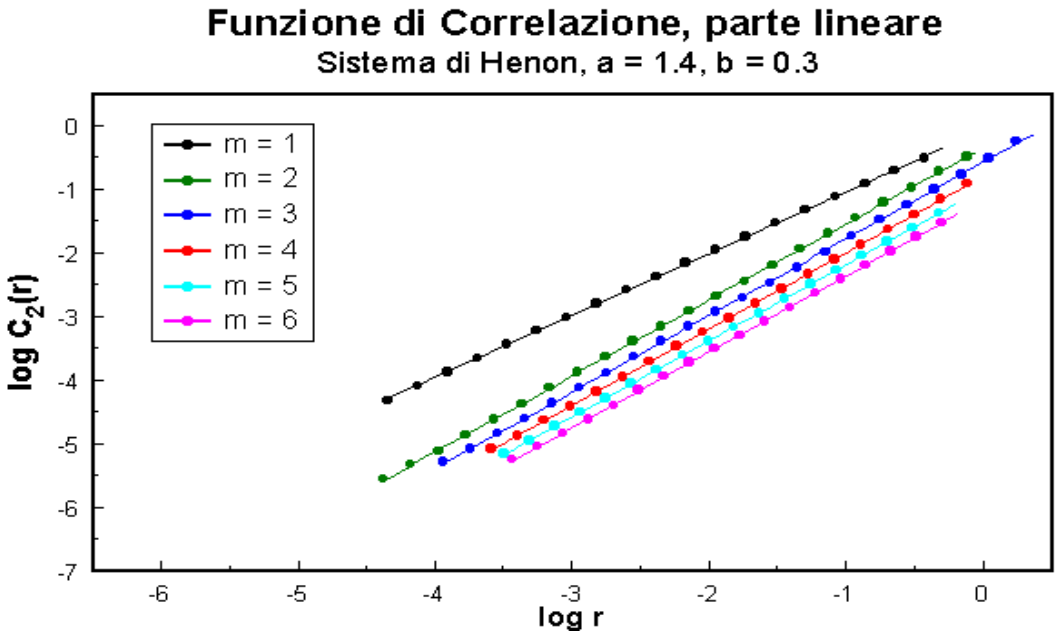

FIGURA 3.7 Zone lineari delle curve di Figura 3.5. Per rendere visibili le rette di interpolazione è stato rappresentato solo un punto ogni tre.

piccola al crescere della dimensione di *embedding*. Tale operazione è soggettiva ed arbitraria e quindi, soprattutto quando si analizzano dati sperimentali può fornire risultati tutt'altro che univoci in relazione alla quota di rumore presente ed al numero dei dati disponibili.

Nelle Figure 3.6 e 3.7 sono rappresentate le zone lineari delle curve delle Figure 3.4 e 3.5 (è stato disegnato solo un dato ogni tre affinché fossero visibili le rette di interpolazione). È evidente come, nel caso del sistema di Lorenz, la pendenza aumenti con la dimensione di *embedding* per $1 \le m \le 3$, per $m \ge 3$ la pendenza rimane pressoché costante ad indicare così che la corretta dimensione di *embedding* è stata raggiunta. Analogamente, per il sistema di Hénon, la pendenza delle rette aumenta quando *m* passa da 1 a 2 per poi rimanere costante per valori di *m* compresi tra 2 e 6.

In base alla (3.21) il valore di saturazione della pendenza corrisponde alla dimensione di correlazione D_2.

Nelle Figure 3.8 e 3.9 sono rappresentate le pendenze delle rette delle Figura 3.6 e 3.7 in funzione della dimensione di embedding. Le dimensioni di correlazione sono state calcolate come la media delle pendenze relative ai valori di saturazione di *m* e sono rappresentate dalle rette orizzontali.

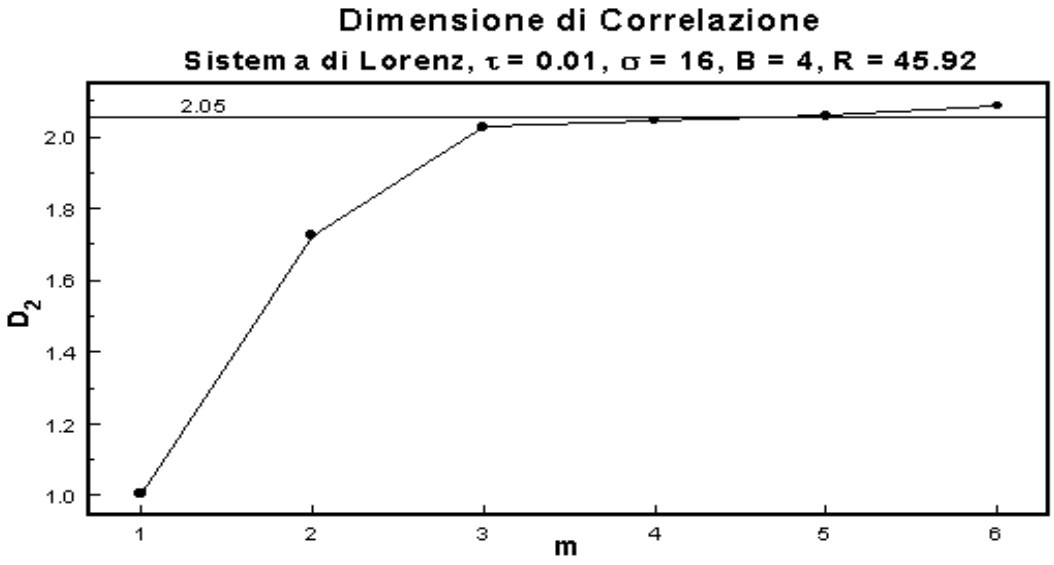

FIGURA 3.8 Pendenza delle rette di Figura 3.6 in funzione della dimensione di *embedding*. La linea orizzontale corrisponde alla media delle pendenze relative ai valori di *m* compresi tra 3 e 6.

Infine, nella tabella 3.1 si confrontano i risultati ottenuti con i valori pubblicati in letteratura (Grassberger e Procaccia, 1983).

I buoni risultati ottenuti nel caso del sistema di Lorenz e del sistema di Hénon non devono far dimenticare che questo metodo, oltre che dipendere da valutazioni di

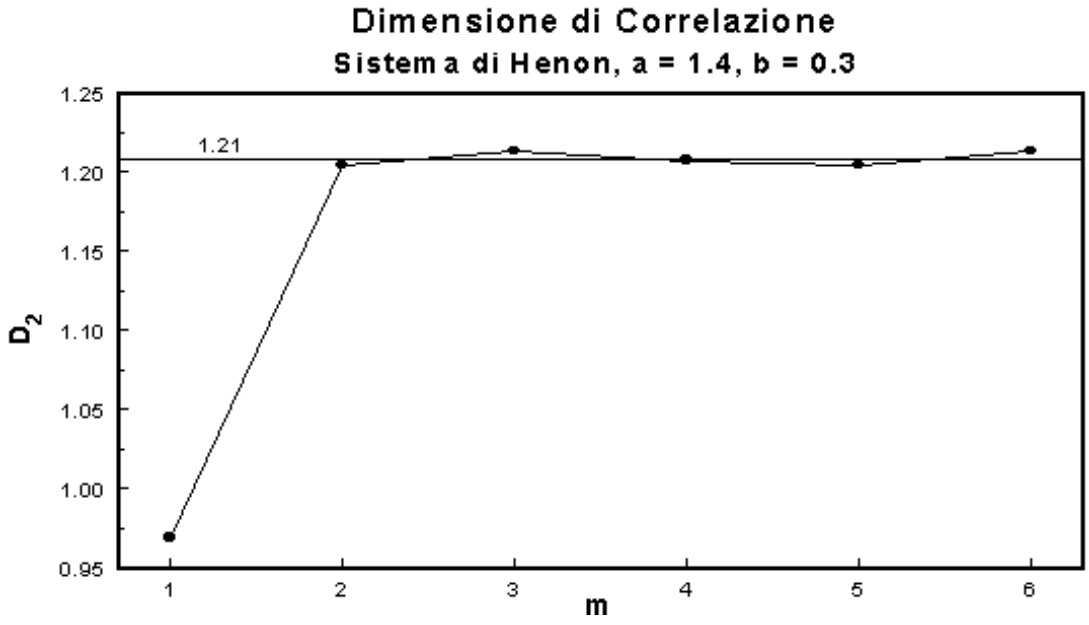

FIGURA 3.9 Pendenza delle rette di Figura 3.7 in funzione della dimensione di *embedding*. La linea orizzontale corrisponde alla media delle pendenze relative ai valori di *m* compresi tra 2 e 6.

DIMENSIONE DI CORRELAZIONE		
	Valori calcolati	**Grassberger e Procaccia**
Sistema di Lorenz	2.05 ± 0.01	2.05 ± 0.01
Sistema di Hénon	1.21 ± 0.02	1.25 ± 0.02

TABELLA 3.1 Confronto tra i valori della dimensione di correlazione calcolati e quelli determinati da Grassberger e Procaccia.

carattere soggettivo, non è del tutto affidabile essendo molto sensibile al rumore cui sono affetti i dati sperimentali. Inoltre sono noti dei casi di saturazione delle pendenze anche in presenza di sistemi stocastici (Theiler, 1986; Osborne e Provenzale, 1989; Provenzale et al., 1992). Per ovviare a quest'ultimo problema si deve ricorrere al metodo della serie surrogata che sarà descritto nel prossimo capitolo.

In generale quindi, a meno che non sia necessario conoscere la dimensione di correlazione, i metodi più indicati per stabilire la caoticità di un sistema sono quello dei falsi vicini, illustrato nel paragrafo 2.3, ed il calcolo del primo esponente di Lyapunov, oggetto dei prossimi paragrafi.

3.3 Gli esponenti di Lyapunov

Come abbiamo visto[9], in un sistema caotico orbite passanti per punti anche molto vicini divergono a causa della dipendenza sensibile dalle condizioni iniziali; la misura della velocità di divergenza delle orbite è nota come *primo esponente di Lyapunov* del sistema.

Per capire il significato geometrico degli esponenti di Lyapunov si considerino gli effetti della dinamica su di un piccolo ipervolume sferico di riferimento contenuto nello spazio delle fasi. Equazioni della dinamica non lineari e generalmente complicate, quali quelle associate a sistemi caotici, avranno l'effetto di far evolvere l'elemento sferico di riferimento verso forme molto complesse. Se però si considerano scale lineari e temporali sufficientemente piccole, l'effetto della

[9] Cfr. Par. 1.2.

dinamica sarà quello di distorcere la sfera in un ellissoide comprimendola lungo alcune direzioni e stirandola lungo altre. L'asse maggiore dell'ellissoide corrisponderà alla direzione più instabile del flusso e il maggiore (o primo) esponente di Lyapunov è la misura del tasso di espansione di tale asse. Più precisamente, indicato con $r(0)$ il raggio iniziale della sfera e con $l_i(t)$ la lunghezza dell'i-esimo asse principale all'istante *t*, l'i-esimo esponente di Lyapunov è definito come:

$$\lambda_i = \lim_{t\to\infty} \ln \frac{l_i(t)}{r(0)}. \tag{3.23}$$

Per convenzione gli esponenti di Lyapunov sono sempre ordinati in modo tale che $\lambda_1 \geq \lambda_2 \geq \lambda_3 \cdots$.

La somma di tutti gli esponenti di Lyapunov misura dunque la velocità di contrazione di un elemento di volume nello spazio delle fasi all'evolvere del tempo e sarà pertanto nulla per i sistemi conservativi e negativa per quelli dissipativi, siano essi caotici o no.

Mentre in un sistema dissipativo stabile il volume tende semplicemente a contrarsi, e quindi gli esponenti sono negativi o nulli, in un sistema dissipativo caotico si ha un compromesso tra due opposte tendenze: la dissipazione ha un ruolo stabilizzante poiché fa contrarre il volume nello spazio delle fasi, mentre l'instabilità orbitale locale fa si che traiettorie inizialmente vicine si separino esponenzialmente e quindi almeno uno degli esponenti deve essere positivo. Pertanto, la presenza di un esponente positivo serve a stabilire se un sistema è caotico.

La divergenza delle traiettorie implica che se le condizioni iniziali di un sistema vengono determinate con un errore E_0, nel corso delle successive iterazioni tale errore verrà amplificato. Una definizione alternativa ed equivalente del primo esponente di Lyapunov serve a caratterizzare la crescita logaritmica media dell'errore relativo per iterazione:

$$\lambda = \lim_{n\to\infty} \lim_{E_0\to 0} \frac{1}{n} \sum_{k=1}^{n} \ln \left| \frac{E_k}{E_{k-1}} \right|. \tag{3.24}$$

In altre parole, un piccolo errore nelle condizioni iniziali sarà amplificato in media di un fattore e^{λ} per ciascuna iterazione.

3.4 Determinazione analitica degli esponenti di Lyapunov

Nel caso in cui sia nota la dinamica del sistema:

$$\mathbf{y}(k) \to \mathbf{F}(\mathbf{y}(k)) = \mathbf{y}(k+1), \tag{3.25}$$

gli esponenti di Lyapunov possono essere determinati osservando l'andamento dell'evoluzione di una piccola perturbazione $\boldsymbol{\delta}(k)$ di un'orbita $\mathbf{y}(k)$.

In base alla (3.25) si ha:

$$\mathbf{y}(k+1)+\boldsymbol{\delta}(k+1) = \mathbf{F}(\mathbf{y}(k)+\boldsymbol{\delta}(k)) \approx \mathbf{DF}(\mathbf{y}(k))\cdot\boldsymbol{\delta}(k)+\mathbf{F}(\mathbf{y}(k))$$

cioè,

$$\boldsymbol{\delta}(k+1) = \mathbf{DF}(y(k))\cdot\boldsymbol{\delta}(k), \tag{3.26}$$

purché $\boldsymbol{\delta}$ rimanga abbastanza piccolo. Nella (3.26) **DF** è la matrice jacobiana:

$$\mathbf{DF}(\mathbf{x})_{ij} = \frac{\partial F_i(\mathbf{x})}{\partial x_j}$$

Passando dall'istante k all'istante $k+L$, la perturbazione subisce un'evoluzione definita da:

$$\begin{aligned}\boldsymbol{\delta}(k+L) &= \mathbf{DF}(\mathbf{y}(k+L-1))\cdot\mathbf{DF}(\mathbf{y}(k+L-2))\cdot\ldots\cdot\mathbf{DF}(\mathbf{y}(k))\cdot\boldsymbol{\delta}(k)\\ &= \mathbf{DF}^L(\mathbf{y}(k))\cdot\boldsymbol{\delta}(k),\end{aligned}$$

ove $\mathbf{DF}^L(\mathbf{x})$ rappresenta la composizione degli L Jacobiani.

Si consideri ora il quadrato della lunghezza del vettore $\boldsymbol{\delta}(k+L)$:

$$\left|\boldsymbol{\delta}(k+L)\right|^2 = \boldsymbol{\delta}^T(k)\cdot\left[\mathbf{DF}^L(\mathbf{y}(k))\right]^T\cdot\mathbf{DF}^L(\mathbf{y}(k))\cdot\boldsymbol{\delta}(k) \tag{3.27}$$

ove l'apice T sta ad indicare la matrice trasposta. La grandezza da determinare nella (3.27) è la matrice:

$$\left[\mathbf{DF}^L(\mathbf{x})\right]^T \cdot \mathbf{DF}^L(\mathbf{x}).$$

In base al *teorema ergodico moltiplicativo* del matematico russo Oseledec quando **x** è un'orbita di un sistema caotico dipendente dal tempo, la matrice:

$$\mathbf{OSL}(\mathbf{x}, L) = \left(\left[\mathbf{DF}^L(\mathbf{x})\right]^T \cdot \mathbf{DF}^L(\mathbf{x})\right)^{\frac{1}{2L}}$$

è tale che la matrice:

$$\mathbf{OSL}(\mathbf{x}) = \lim_{L\to\infty} \mathbf{OSL}(\mathbf{x}, L)$$

esiste ed è indipendente da **x** per ogni **x** appartenente al bacino di attrazione dell'attrattore cui appartiene l'orbita; inoltre i logaritmi degli autovalori di tale matrice ortogonale sono gli esponenti di Lyapunov λ_i, $i = 1,2,\ldots,d$ del sistema dinamico $\mathbf{x} \to \mathbf{F}(\mathbf{x})$.

Di fatto la determinazione degli autovalori della matrice di Oseledec è tutt'altro che banale; tale problema richiede l'utilizzo di una decomposizione ricorsiva *QR* la cui descrizione esula dagli scopi di questa tesi.

3.5 Calcolo degli esponenti di Lyapunov dai dati sperimentali

Per determinare gli esponenti di Lyapunov da una serie temporale di dati sperimentali utilizzando il procedimento descritto nel paragrafo precedente occorre un metodo che permetta di calcolare la matrice Jacobiana $\mathbf{DF}(\mathbf{y}(k))$.

Dato il punto $\mathbf{y}(k)$, si trovano i suoi N_B primi vicini: $\mathbf{y}^{(r)}(k)$, $r = 1,2,\ldots,N_B$. Ciascun vicino evolve in un punto noto: $\mathbf{y}^{(r)}(k) \to \mathbf{y}(r; k+1)$ che sarà in prossimità di $\mathbf{y}(k+1)$ (la notazione serve a distinguere $\mathbf{y}(r; k+1)$, successivo di $\mathbf{y}^{(r)}(k)$, da $\mathbf{y}^{(r)}(k+1)$, r-esimo primo vicino di $\mathbf{y}(k+1)$). Se si definisce una mappa locale che a ciascun vicino associa il suo successivo:

$$\mathbf{y}(r;k+1)=\sum_{m=1}^{M}\mathbf{c}(m,k)\phi_m(\mathbf{y}^{(r)}(k))$$

ove M è la dimensione dello spazio, le funzioni $\phi_m(\mathbf{x})$ formano un sistema di base scelto a priori e i coefficienti locali $\mathbf{c}(m,k)$ sono determinati minimizzando i residui

$$\sum_{r=1}^{N_B}\left|\mathbf{y}(r;k+1)-\sum_{m=1}^{M}\mathbf{c}(m,k)\phi_m(\mathbf{y}^{(r)}(k))\right|^2,$$

allora le componenti della matrice Jacobiana sono:

$$DF_{ij}(\mathbf{y}(k))=\sum_{m=1}^{M}c_i(m,k)\frac{\partial\phi_m(\mathbf{y}(k))}{\partial\, y_j(k)}.$$

La matrice Jacobiana così trovata serve a determinare la matrice di Oseledec[10] che deve essere diagonalizzata con la decomposizione ricorsiva *QR*.

Come si vede questo metodo è oltremodo complesso e richiede un uso particolarmente intensivo delle risorse di calcolo; inoltre i risultati non sono molto affidabili se si dispone di serie sperimentali brevi. D'altro canto per stabilire se un sistema è caotico è sufficiente stimare l'esponente di Lyapunov maggiore (λ_1)[11]. Fortunatamente esiste un metodo per il calcolo di λ_1 proposto da Sato (Sato et al., 1987) e perfezionato da Rosenstein (Rosenstein et al., 1993) che consente di superare le difficoltà cui si accennava più sopra.

Il metodo di Rosenstein si basa sul fatto che due punti vicini nell'attrattore diano origine a due traiettorie che divergono esponenzialmente con esponente pari all'esponente positivo maggiore di Lyapunov. Se $d(t)$ rappresenta la divergenza media di due traiettorie al tempo *t*, si avrà allora:

$$d(t)=Ce^{\lambda t} \qquad (3.28)$$

con *C* costante e $\lambda\equiv\lambda_1$.

[10] Si veda il paragrafo precedente.
[11] Cfr. par. 3.3.

Dato dunque un punto dell'attrattore $\mathbf{y}(k)$, ed il suo primo vicino $\mathbf{y}^{(n)}(k)$, sia $d_k(0)$ la loro distanza. Dopo un tempo $t = j\tau$, dove τ rappresenta il tempo di campionamento e $j = 1,2,\ldots$; in base alla (3.28) la loro distanza sarà:

$$d_k(j\tau) = C_k e^{\lambda j\tau}, \tag{3.29}$$

ove la costante C_k rappresenta la distanza iniziale.

Passando ai logaritmi al (3.29) diviene:

$$\ln d_k(j\tau) = \ln C_k + \lambda j\tau. \tag{3.30}$$

La (3.30) rappresenta un insieme di rette, una per ciascun punto iniziale $\mathbf{y}(k)$. L'esponente può allora essere calcolato attraverso un *fit* della linea media definita come:

$$f(j) = \frac{1}{M}\sum_{k=1}^{M} \ln d_k(j\tau), \tag{3.31}$$

dove M è il numero di punti sull'attrattore.

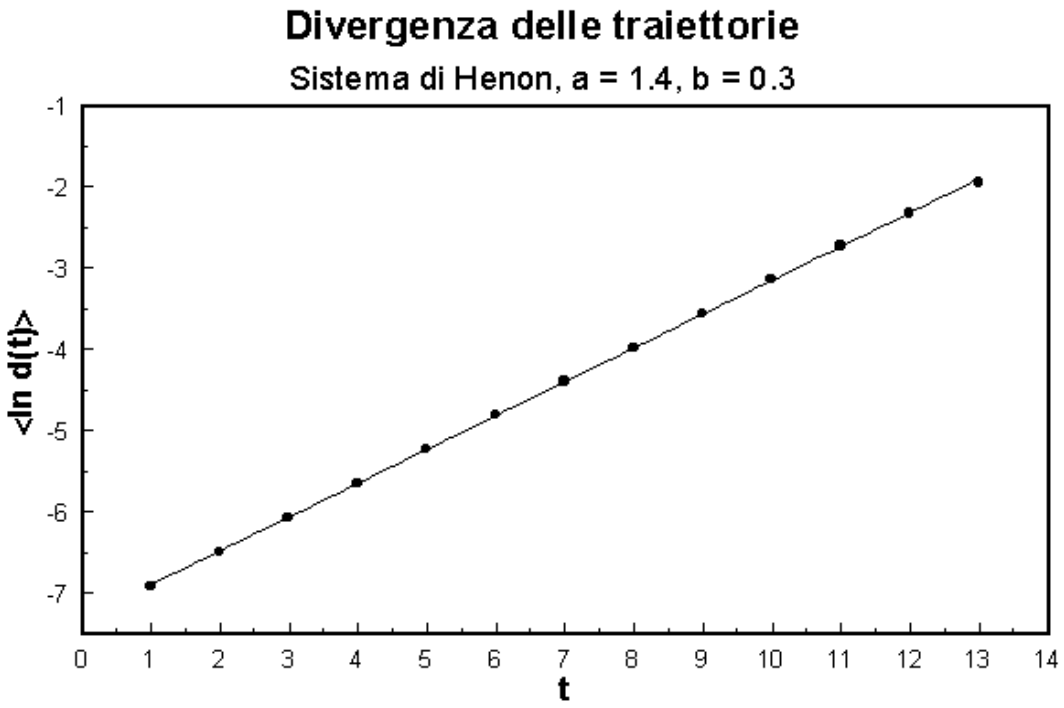

FIGURA 3.10 Divergenza media delle traiettorie per il sistema di Hénon.

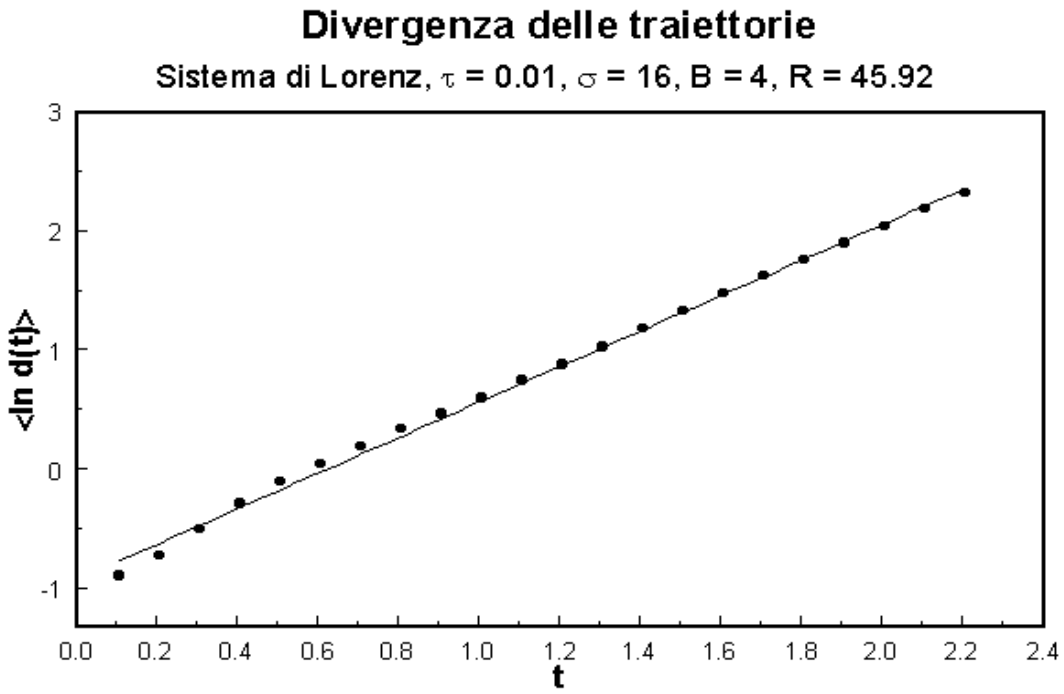

FIGURA 3.11 Divergenza media delle traiettorie per il sistema di Lorenz.

La media nell'equazione (3.31) è la chiave per calcolare valori accurati di λ anche utilizzando serie temporali corte e rumorose.

Nelle figure 3.10 e 3.11 sono rappresentate le rette di equazione (3.31) per il sistema di Hénon e per quello di Lorenz. I valori di λ che si desumono dalle pendenze delle rette sono confrontati in Tabella 3.2 con valori che si trovano in letteratura. Come si vede il metodo di Rosenstein consente di ottenere risultati molto buoni.

ESPONENTE DI LYAPUNOV		
	Valori calcolati	**Valori pubblicati**
Sistema di Hénon	0.416	0.419
Sistema di Lorenz	1.490	1.502

Tabella 3.2 Confronto dei valori calcolati dell'esponente di Lyapunov con valori pubblicati in letteratura. Il dato relativo al sistema di Hénon si trova in (Peitgen et al., 1992), quello relativo al sistema di Lorenz si trova in (Abarbanel et al., 1993).

4. I dati sperimentali

4.1 Introduzione

La serie esaminata, (Ceschia, 1998), è relativa alle temperature registrate alle ore 9 presso l'Osservatorio Malignani a Udine-Castello. Le temperature, espresse in gradi Celsius, sono state rilevate con la precisione strumentale di un decimo di grado. La serie è stata sottoposta ad accurati test di consistenza ed a verifiche di omogeneità di misurazione nel corso degli anni.

La serie di osservazioni meteorologiche, iniziata a Udine-Castello ad opera di A. Malignani nel 1894, continua ormai da oltre cent'anni ad opera di tre generazioni della stessa famiglia.

La stazione è situata sul lato Sud-Ovest del colle del Castello di Udine, alla quota di 136 m sul livello del mare. I termometri a mercurio erano difesi "*contro l'irradiazione solare e notturna da una torre e una tettoia*" (Malignani, 1939). Ora la vecchia stazione storica è stata trasformata in una stazione automatica dotata dei più moderni sensori ed è gestita in collaborazione con il Dipartimento di Fisica dell'Università di Udine.

4.1 Introduzione

Nonostante la cura con cui i membri della Famiglia Malignani hanno seguito nel corso degli anni la raccolta dei dati dell'Osservatorio, la serie presenta alcuni buchi prodotti per lo più da cause di forza maggiore. La maggior parte dei dati mancanti, infatti, corrisponde al periodo che va dall'ottobre del 1917 all'inizio del 1919, ossia tra la disfatta di Caporetto e la fine della prima guerra mondiale.

Nel presente lavoro ci si è limitati ad utilizzare il più lungo spezzone completo della serie che va dal 1° febbraio 1920 al 31 dicembre 1996. Le principali caratteristiche della serie sono riassunte nella Tabella 4.1; la serie è rappresentata nella Figura 4.1.

Ora di rilevazione	**9:00**
Periodo di campionamento	**24 h**
Quantità dei dati	**28094**
Minimo	**-13.5**
Massimo	**32.0**
Media	**12.35**
Deviazione standard	**7.94**

TABELLA 4.1. Principali caratteristiche della serie studiata.

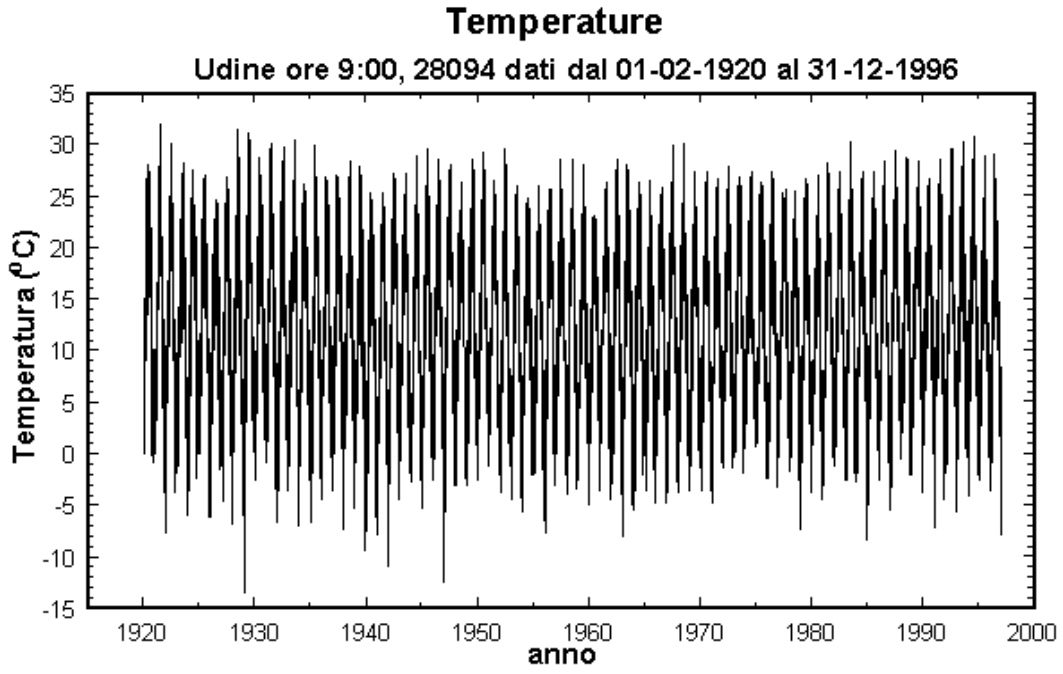

FIGURA 4.1 La serie di temperature di Udine-Castello.

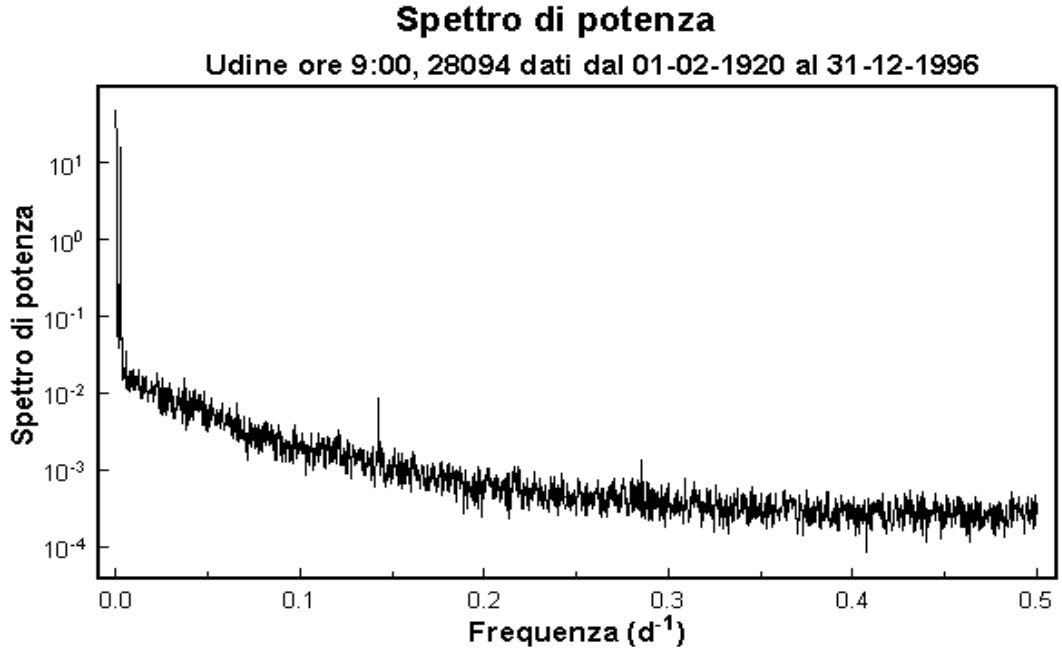

FIGURA 4.2 Spettro di potenza della serie di temperature.

Nella Figura 4.2 è rappresentato lo spettro di potenza della serie in esame. Nell'ingrandimento della zona a bassa frequenza di Figura 4.3 si vede come, a parte il valore massimo relativo alla componente a frequenza nulla (costante), sia presente il picco relativo all'andamento annuale.

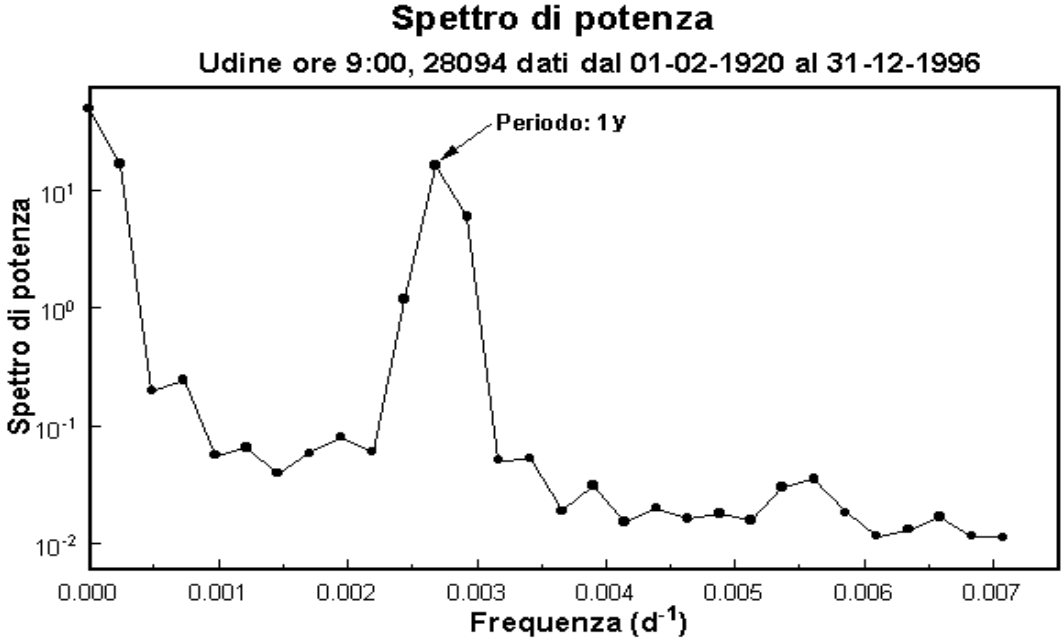

FIGURA 4.3 Ingrandimento della zona a bassa frequenza dello spettro di Figura 4.2.

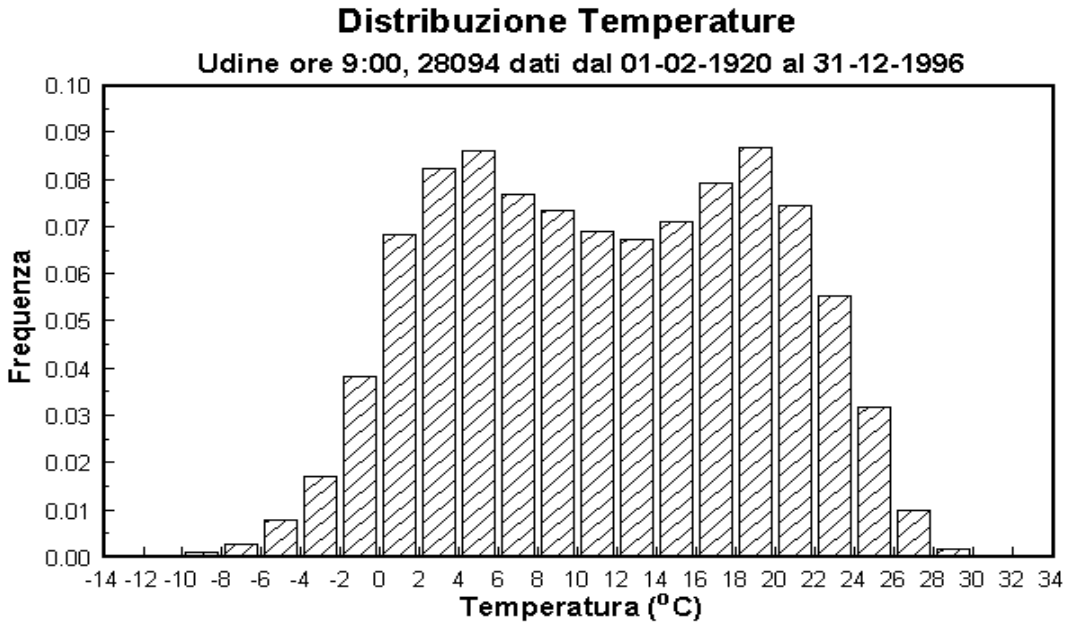

FIGURA 4.4 Distribuzione della serie di temperature.

Nelle Figure 4.4 e 4.5 sono rappresentate rispettivamente la distribuzione delle temperature e l'autocorrelazione della serie.

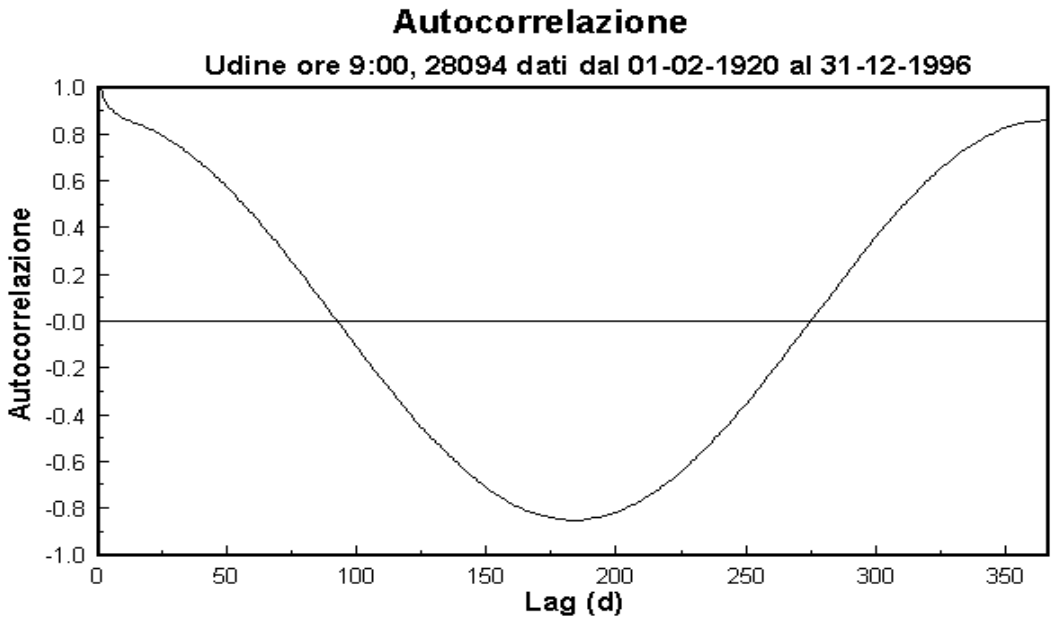

FIGURA 4.5 Autocorrelazione della serie di temperature.

4.2 Denoising: il metodo delle wavelet

Una serie temporale può possedere caratteristiche caotiche le quali possono essere nascoste dalla presenza di rumore. Se l'intensità del rumore è sufficientemente alta, il metodo di Takens non consente la ricostruzione corretta dell'attrattore. In generale il rumore presente nelle serie sperimentali possiede una dimensione molto maggiore di quella del sistema dinamico soggiacente e pertanto può mascherarne le caratteristiche (Cuomo et al., 1993). Diversi sono i tentativi di sviluppare metodi pratici per eliminare il rumore presente in una serie (Sauer, 1992; Grassberger et al., 1993; Kostelich and Schreiber, 1993). Alcuni di questi metodi (Donoho, 1993; Saito, 1994) si basano sulla recente tecnica di analisi e sintesi dei dati nota come *Wavelet Transform* (Daubechies, 1988; Mallat, 1989; Kaiser, 1994) che è stata adottata anche in questo lavoro.

4.2.1 Continuous Wavelet Transform

La *continuous wavelet transform* di un segnale $f(t)$ è definita dalla relazione:

$$\tilde{f}(a,b) = \int_{-\infty}^{+\infty} \Psi^*_{a,b}(t)\, f(t) dt \qquad (4.1)$$

ove

$$\Psi_{a,b}(t) \equiv \mid a \mid^{-p} \Psi\left(\frac{t-b}{a}\right) \qquad (4.2)$$

è una funzione a valori complessi e media nulla appartenente a $L^2(\mathbf{R})$; il valore di p è irrilevante e ci si atterrà alla convenzione prevalente ponendo $p = 1/2$ (Kaiser, 1994).

La scelta di Ψ è largamente arbitraria e per il presente lavoro è stata considerata la *Mexican hat function* (Figura 4.6) che è l'opposto della derivata seconda della distribuzione normale a media nulla e varianza unitaria:

$$\Psi(t) = \frac{(1-t^2)e^{-t^2/2}}{\sqrt{2\pi}}. \qquad (4.3)$$

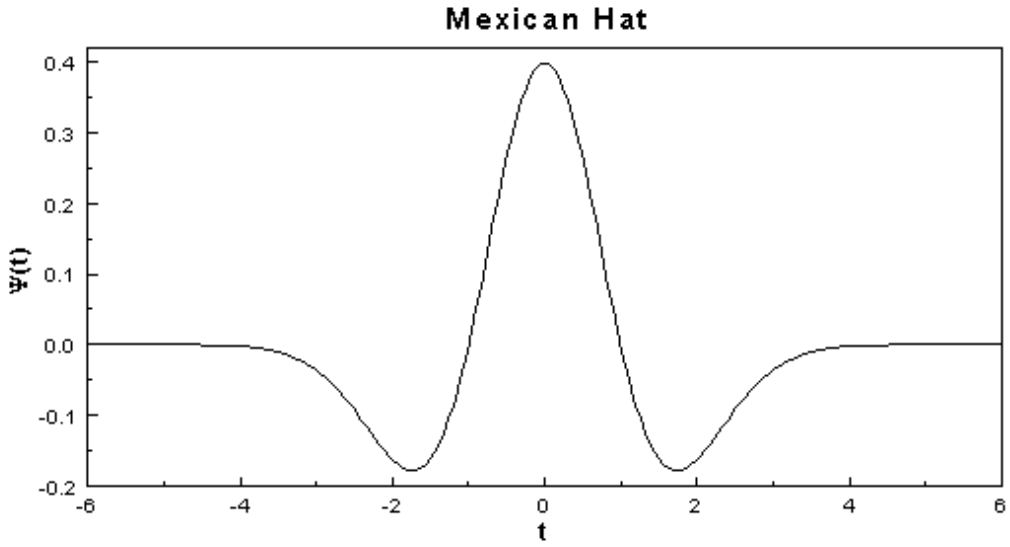

FIGURA 4 6 Diagramma della funzione di equazione 4.3.

Si dimostra (Kaiser, 1994) che, in tal caso, $\tilde{f}(a,b)$ rappresenta il dettaglio di secondo ordine di *f* alla scala *a*, nel senso che $\tilde{f}(a,b)$ è la derivata seconda di una "media mobile" di *f* calcolata mediante una gaussiana traslata (parametro *b*) e dilatata (parametro *a*).

In altre parole, un dato coefficiente della trasformata, ossia un dato elemento $\tilde{f}(a_0,b_0)$ di $\tilde{f}(a,b)$, sarà in modulo tanto maggiore quanto più il segnale risulti variabile in un intorno di b_0, con un "periodo" intorno a $4a_0$.

L'antitrasformata della (4.1) è data dalla:

$$f_1(t) = \int_0^{+\infty} a^{2p-3} da \int_{-\infty}^{+\infty} \Psi_{a,b}(t)\, \tilde{f}(a,b) db, \qquad (4.4)$$

in effetti la $f_1(t)$ differisce dal segnale originale *f* nel senso che il procedimento di analisi e sintesi sottrae ad ogni valore di $f_1(t)$ la media locale di *f* calcolata intorno a *t*.

Il ruolo giocato dal parametro *a* è dunque quello di analizzare il segnale a diverse scale della variabile temporale (a piccoli valori di *a* corrispondono scale più fini); nell'ipotesi quindi che il segnale sia affetto da rumore ad alta frequenza, risulta evidente come, sostituendo al limite inferiore di integrazione della variabile *a* nella

(4.4) una soglia $a_0 > 1$, sia possibile ottenere un segnale depurato dal rumore (*denoising*).

4.2.2. Descrizione dell'algoritmo

L'analisi dei dati è stata implementata mediante integrazione numerica della (4.1) con passo di integrazione unitario, avendo scelto l'unità di tempo pari alla distanza (costante) tra due dati successivi della serie temporale:

$$\tilde{f}(a,b) = \frac{1}{\sqrt{a}} \sum_i \Psi\left(\frac{t_i - b}{a}\right) f(t_i). \qquad (4.5)$$

Per ovviare agli effetti di bordo causati dalla lunghezza finita del segnale si sono imposte condizioni al contorno cicliche: tale scelta, seppur arbitraria, si è rivelata alquanto efficace. Il parametro a varia da 1 ad un valore a_{max} determinato dall'utente, b varia da 1 a b_{max} pari al numero di dati della serie. Nella Figura (4.7) è rappresentata una serie di valori della variabile x del sistema di Lorenz con il 50% di rumore bianco additivo (curva tratteggiata) e la stessa serie priva di rumore (curva

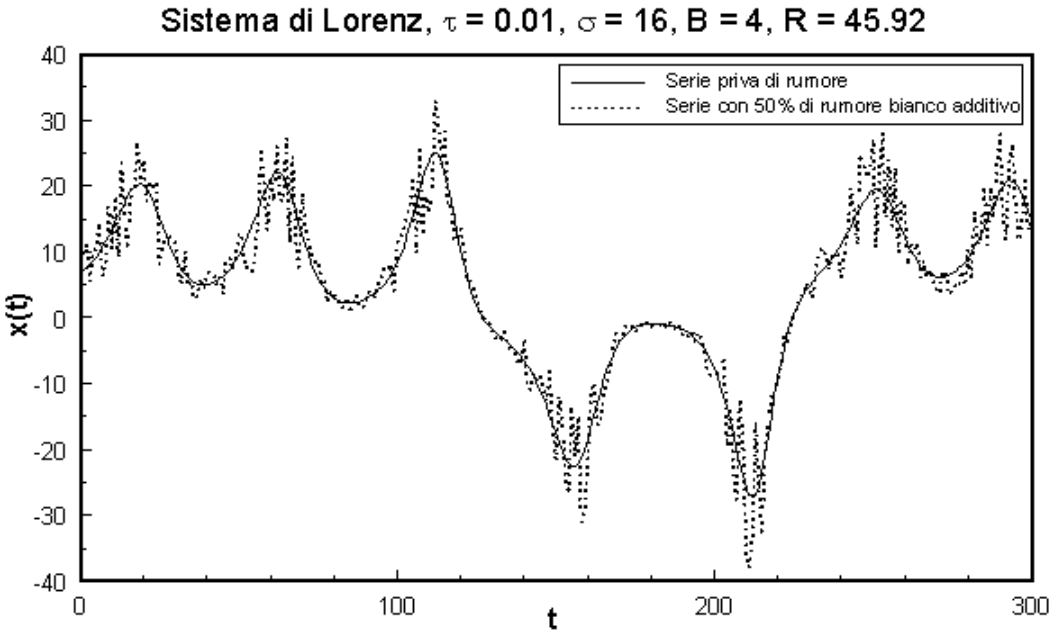

FIGURA 4.7 Variabile x del sistema di Lorenz con 50% di rumore bianco additivo (curva tratteggiata) e la stessa serie priva di rumore (curva continua).

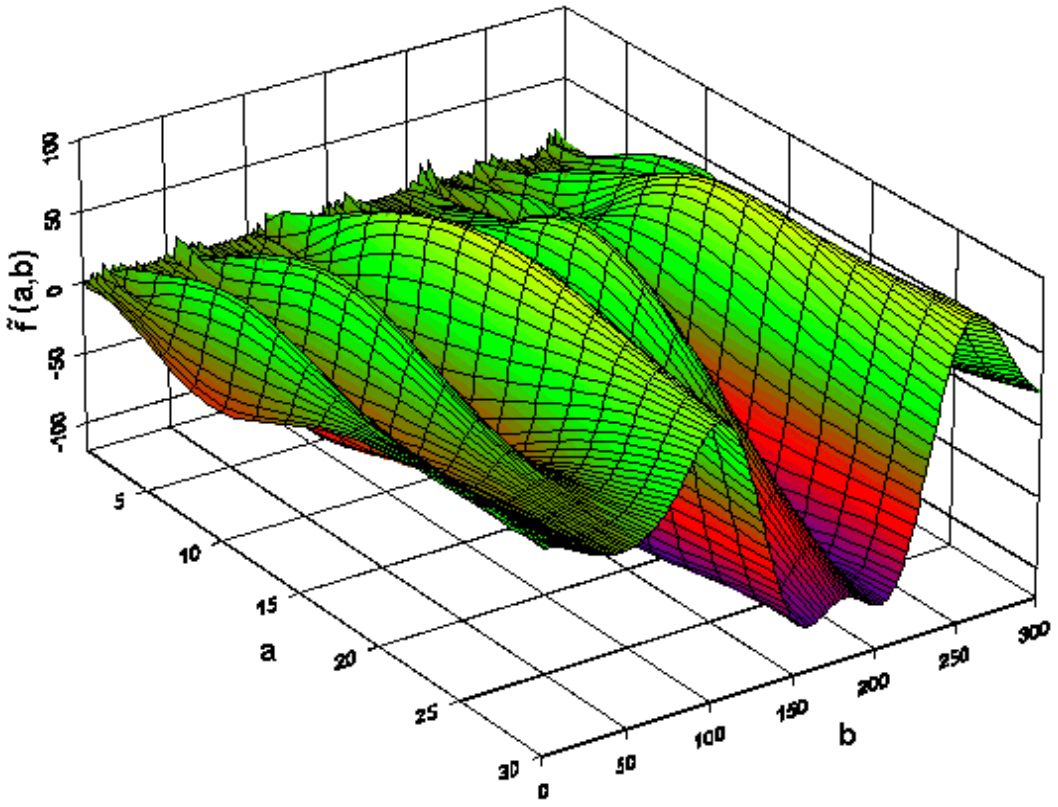

FIGURA 4.8 Superficie di equazione $z = \tilde{f}(a,b)$ (4.5) relativa alla serie rumorosa di Figura (4.7).

continua). La serie di Lorenz rumorosa è stata usata per valutare l'efficacia del metodo. Nella figura (4.8) è rappresentata la superficie di equazione $z = \tilde{f}(a,b)$ relativa alla serie rumorosa di Figura (4.7).

La sintesi (ricostruzione) è stata ottenuta mediante integrazione numerica della (4.4):

$$f_1(t) = \sum_a \sqrt{a} \sum_b \Psi\left(\frac{t_i - b}{a}\right) \tilde{f}(a,b) \qquad (4.6)$$

b varia da 1 a b_{max} come per l'analisi, mentre gli estremi di integrazione per *a* sono forniti in input. Ciò consente, come opzione, di effettuare il denoising del segnale.

Il vettore $f_1(t)$, risultato della sintesi, viene fornito come input ad un secondo modulo il quale aggiunge ad ogni elemento di $f_1(t)$ la media locale calcolata attorno a *t*; questo procedimento viene ripetuto diverse volte per diverse larghezze (*w*) della finestra sulla quale si calcola la media locale. Si ottengono così diversi vettori per

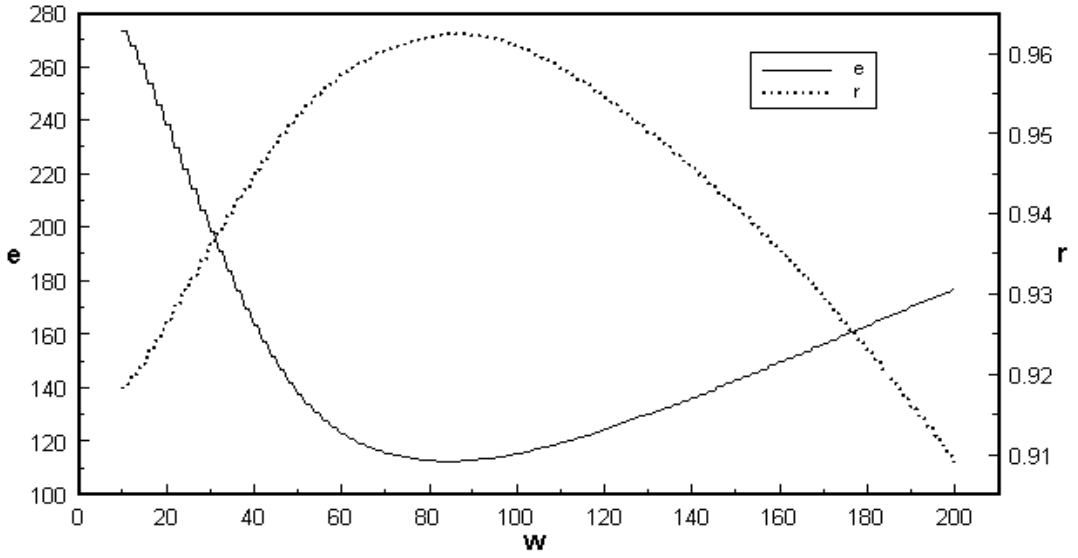

FIGURA 4.9 Errore quadratico medio (*e*) e coefficiente di correlazione (*r*) tra la serie rumorosa originale e quella ricostruita $f_1(t)$ in funzione della larghezza della finestra (*w*) sulla quale si calcola la media mobile aggiunta a ciascun punto di $f_1(t)$.

ciascuno dei quali vengono calcolati il coefficiente di correlazione (*r*) e l'errore quadratico medio (*e*) rispetto al segnale originale.

Nella figura 4.9 sono rappresentati i valori di *r* ed *e* in funzione della larghezza *w* della finestra; è così possibile operare una scelta di *w* tale da rendere minimo l'errore quadratico medio o da rendere massimo il coefficiente di correlazione (non sempre tali condizioni si ottengono in corrispondenza ad un unico valore di *w*). In corrispondenza a tale scelta si ottiene infine il segnale ricostruito sommando ad ogni elemento di $f_1(t)$ la media locale calcolata attorno a *t*.

Il procedimento descritto è stato applicato alla serie rumorosa di Figura 4.7: nell'analisi il parametro *a* assumeva valori compresi tra $a_{\min} = 1$ e $a_{\max} = 30$; il *denoising* è stato ottenuto annullando nella ricostruzione i coefficienti $\tilde{f}(a,b)$ con $a = 1$ e $a = 2$. Come si vede nella Figura 4.9, il valore ottimale della larghezza della finestra sulla quale calcolare la media locale è risultato pari a 84.

Nella Figura 4.10 vengono confrontate la serie *denoised* con quella priva di rumore. La validità del procedimento è confermata dal fatto che l'applicazione del metodo dei falsi vicini alla serie *denoised* consente di determinare la corretta

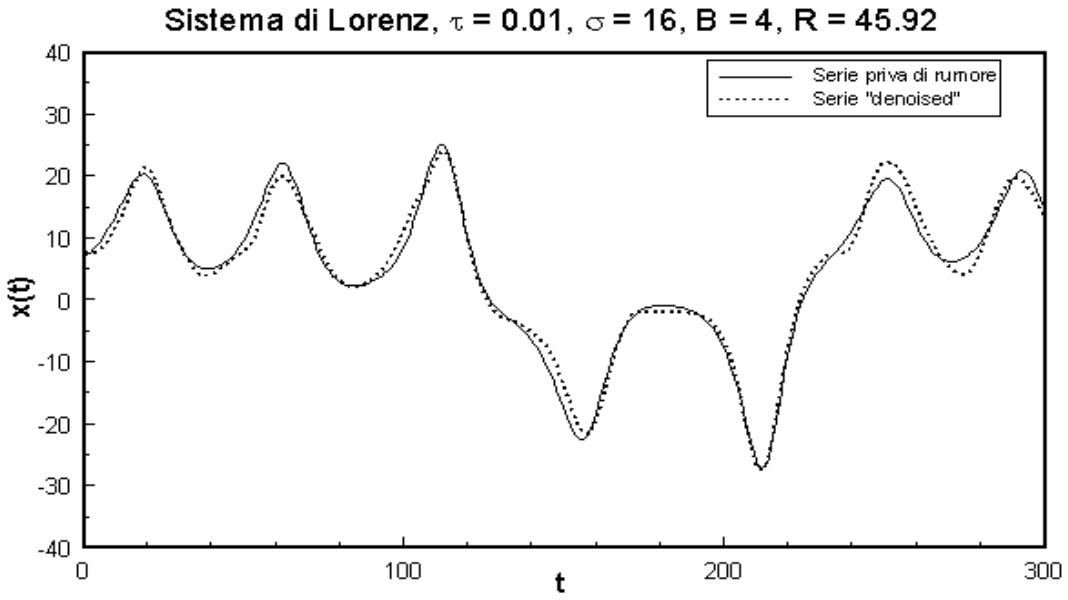

FIGURA 4.10 Confronto tra la serie *denoised* e quella priva di rumore di Figura 4.7.

dimensione di embedding come si vede nella Figura 4.11, anche se il rumore residuo fa sì che il numero di falsi vicini corrispondente a $m = 3$ sia leggermente maggiore di zero anziché nullo.

Il metodo descritto è stato applicato alla serie delle temperature adottando l'approccio più conservativo: il *denoising* è stato ottenuto annullando nella ricostruzione i coefficienti $\tilde{f}(a,b)$ con $a = 1$, ossia solo quelli alla frequenza più

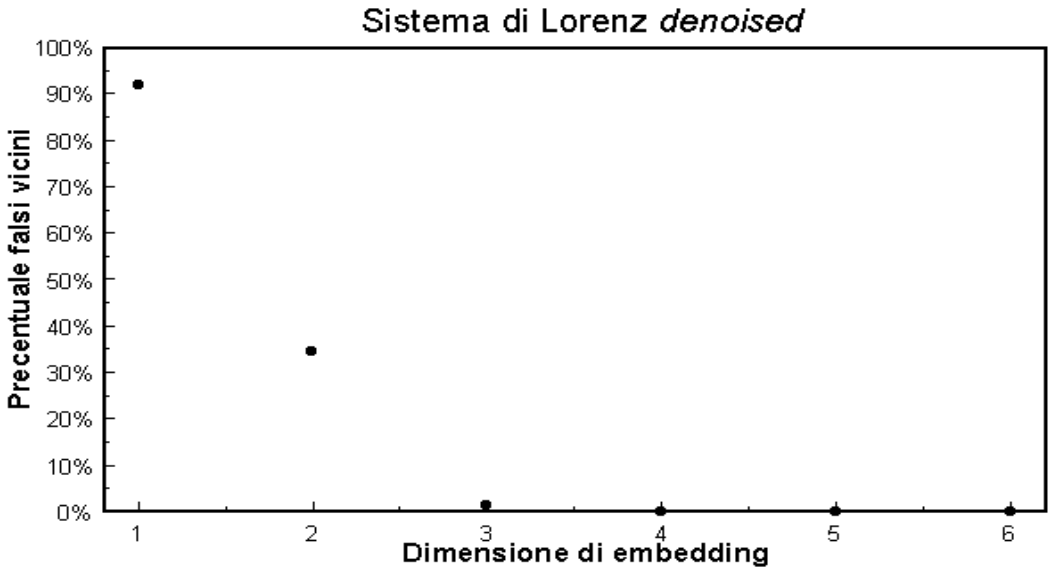

FIGURA 4.11 Falsi vicini per la serie *denoised* di Figura 4.7.

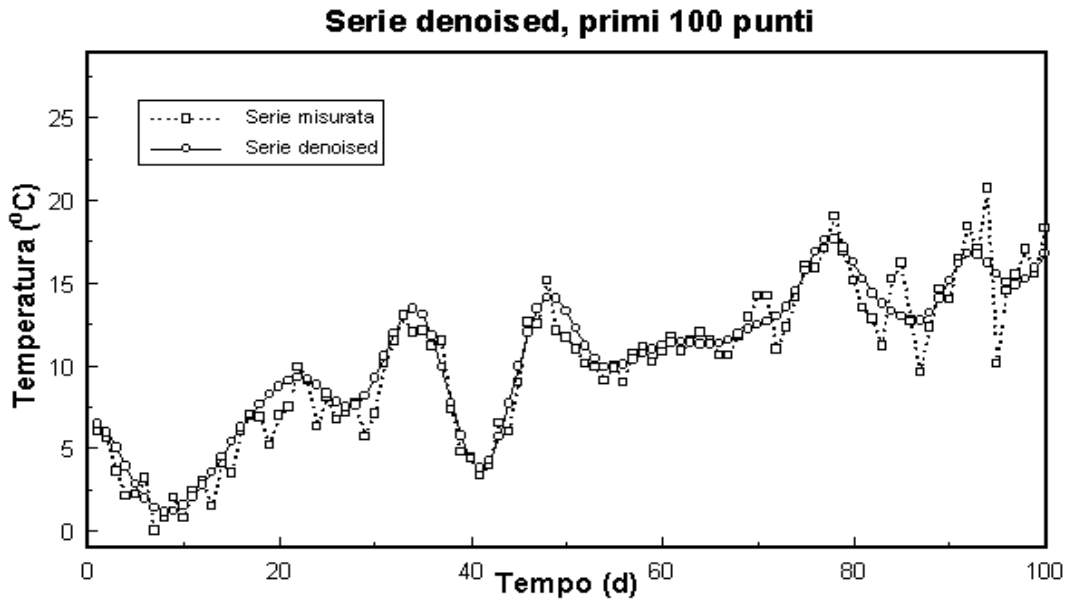

FIGURA 4.12 Confronto tra la serie di temperature misurata e quella *denoised*.

alta.

Nella Figura 4.12 sono rappresentati i primi 100 punti della serie *denoised* e quelli della serie misurata.

Per verificare che la componente sottratta col procedimento di *denoising* è

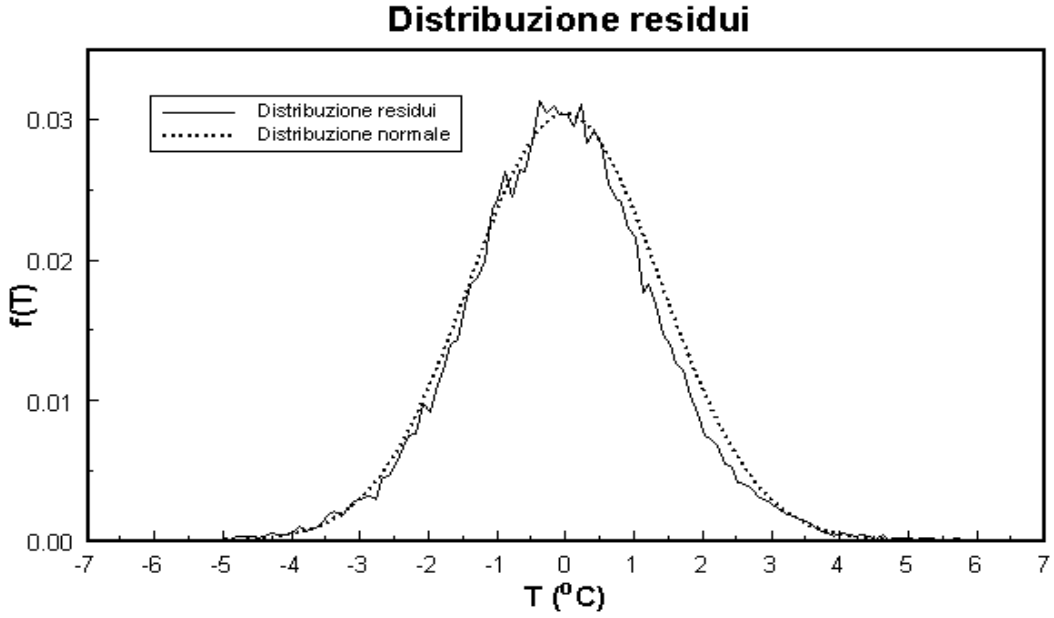

FIGURA 4.13 Confronto tra la distribuzione dei residui e quella normale.

effettivamente un rumore bianco, la distribuzione dei residui (differenze tra la serie misurata e quella *denoised*) è stata confrontata (Figura 4.13) con la distribuzione normale avente la stessa media ($\mu = 0.00$) e varianza ($\sigma^2 = 1.95$) dei residui. Come si vede dalla figura, il buon accordo tra la distribuzione dei residui e quella normale conferma la natura casuale della componente sottratta.

5. Applicazione ai dati sperimentali, un modello di previsione delle temperature

5.1 Ricostruzione dello spazio delle fasi

L'utilizzo del teorema di embedding per ricostruire lo spazio delle fasi relativo alla serie di temperature di Udine-Castello descritta nel capitolo 4, richiede la determinazione del *time delay* e della dimensione di *embedding* (cfr. par. 2.1.). A tal fine sono stati utilizzati i metodi di analisi esposti nel capitolo 2.

5.1.1 Mutua informazione media

Nelle figure 5.1 e 5.2 è rappresentata la mutua informazione media (2.8) per la serie dei dati sperimentali e per la serie *denoised* (cfr. par. 4.2.2.). Come si vede, entrambe le curve non presentano minimi, tuttavia la mutua informazione media della serie *denoised* diminuisce più lentamente e quindi il *time delay* è stato posto uguale a 1 per la serie dei dati sperimentali e uguale a 2 per quella *denoised*.

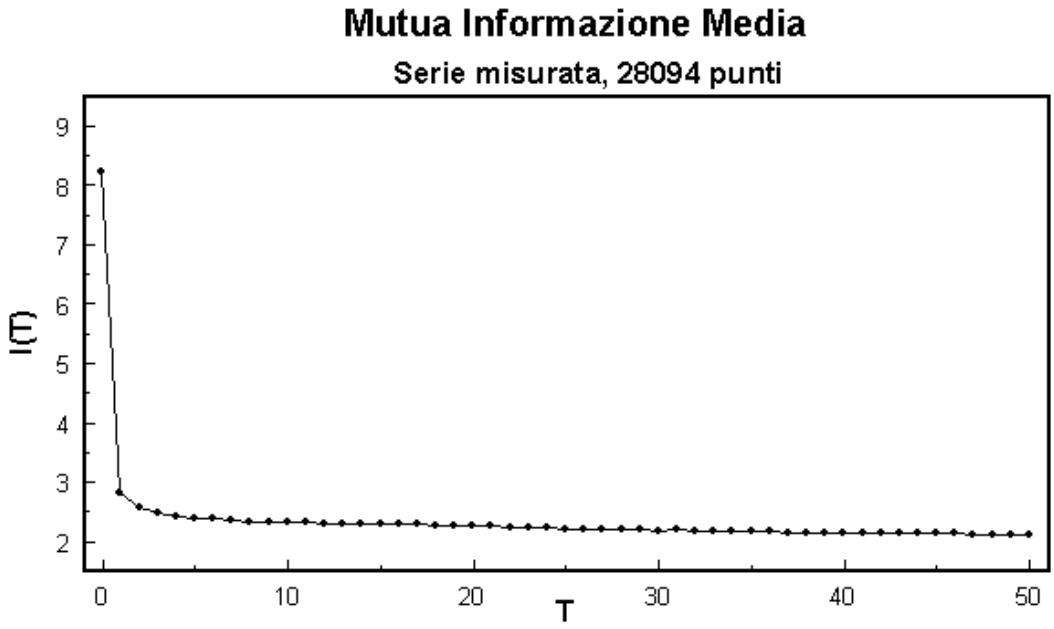

FIGURA 5.1 Mutua informazione media per la serie misurata.

5.1.2 Falsi vicini

Il risultato del calcolo dei falsi vicini per la determinazione della dimensione di embedding è rappresentato in Figura 5.3 per la serie misurata e in Figura 5.4 per quella *denoised*.

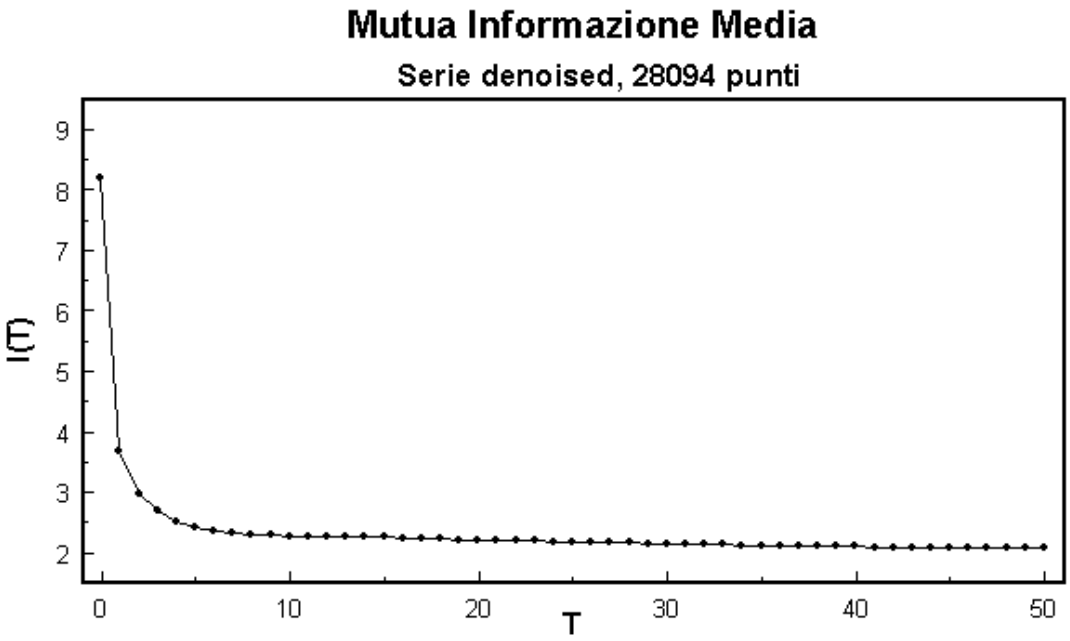

FIGURA 5.2 Mutua informazione media per la serie *denoised*.

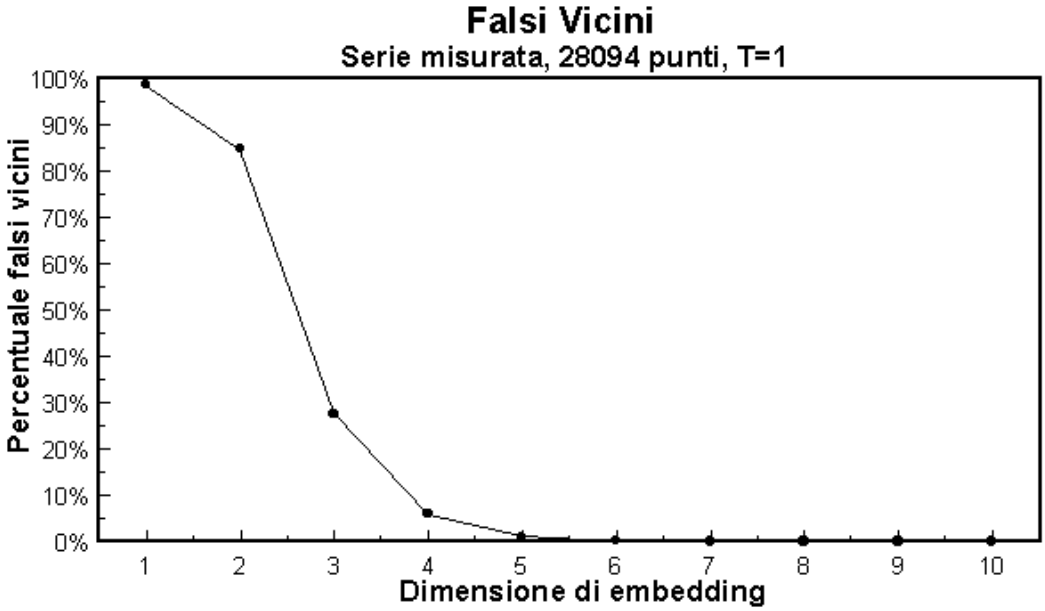

FIGURA 5.3 Falsi vicini per la serie misurata.

Come si vede dalle figure, in entrambi i casi il metodo individua una dimensione di *embedding* pari a 5 anche se, a causa del rumore, la percentuale di falsi vicini è sempre leggermente maggiore nel caso della serie misurata. Questo

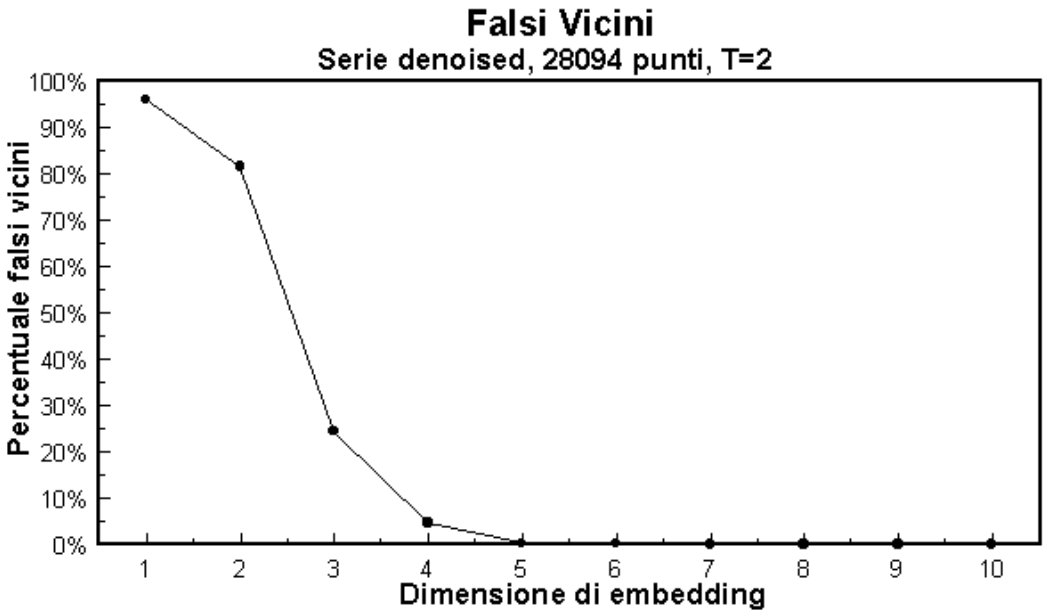

FIGURA 5.4 Falsi vicini per la serie *denoised*.

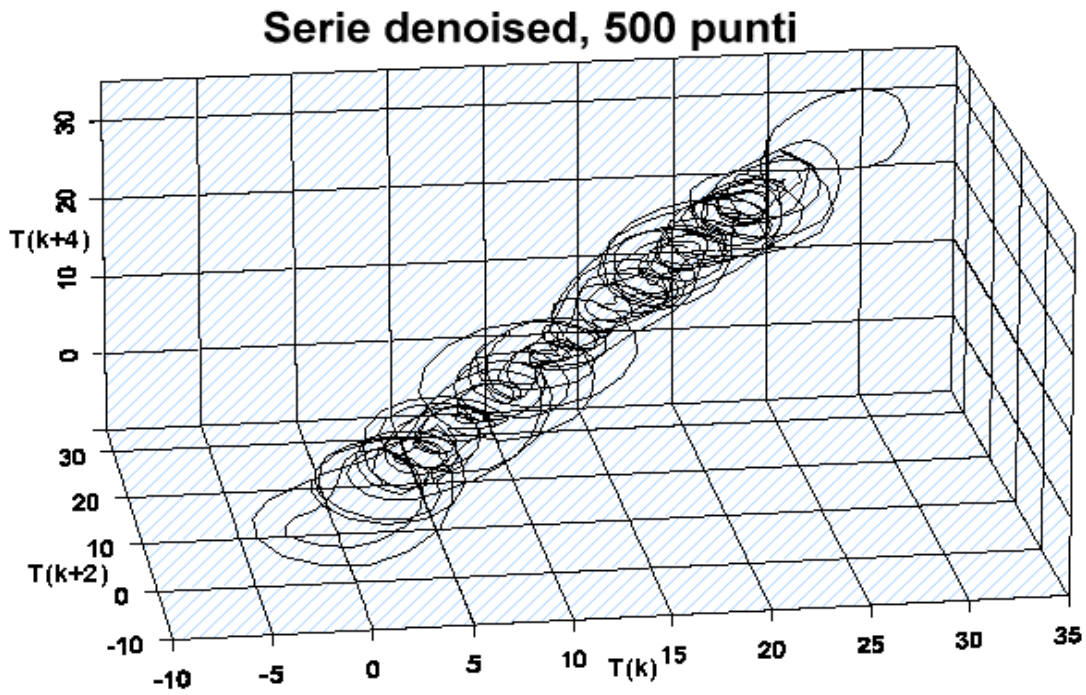

FIGURA 5.5 Sezione tridimensionale dell'attrattore della serie *denoised*.

risultato mostra che il metodo dei falsi vicini è efficace anche se applicato direttamente alla serie rumorosa.

Nella Figura 5.5, infine, è rappresentata una sezione tridimensionale dell'attrattore della serie *denoised* nello spazio delle fasi, ricostruito usando terne di valori della temperatura della forma: (T_k, T_{k+2}, T_{k+4}).

5.2 Invarianti della dinamica

Come già fatto per i sistemi di Lorenz ed Hènon, anche alla serie dei dati delle temperature giornaliere di Udine sono stati applicati gli algoritmi di Grassberger-Procaccia ed è stata effettuata la ricerca del primo esponente di Lyapunov.

5.2.1 Algoritmo di Grassberger-Procaccia

L'applicazione dell'algoritmo di Grassberger-Procaccia sia alla serie misurata che a quella *denoised* ha prodotto risultati ambigui a causa dei problemi illustrati nel paragrafo 3.2. poiché non è stato possibile determinare con sicurezza la zona lineare delle curve. Ciò è probabilmente dovuto a due cause principali: in primo luogo

l'algoritmo richiede il calcolo delle distanze tra tutti i punti dell'attrattore e quindi l'allocazione di un vettore avente un numero di componenti proporzionale a $N^2/2$ ove N è il numero dei punti; le dimensioni di tale vettore sono però limitate dalla memoria a disposizione e quindi nel calcolo della funzione di correlazione non è stato possibile utilizzare più di 10000 punti. L'altra causa è rappresentata dal rumore presente nella serie misurata e da quello residuo della serie *denoised*, evidentemente ancora sufficiente a rendere inefficace l'algoritmo.

D'altro canto l'uso del metodo di Grassberger-Procaccia per determinare la caoticità di una serie non è consigliato (Theiler, 1986; Osborne e Provenzale, 1989; Provenzale et al., 1992) essendo molto più attendibile il metodo dei falsi vicini o il calcolo del primo esponente di Lyapunov; inoltre la dimensione di correlazione, unica informazione aggiuntiva fornita dall'algoritmo di Grassberger-Procaccia, non è necessaria agli scopi del presente lavoro.

5.2.2. Esponente di Lyapunov

I risultati dell'applicazione del metodo di Rosenstein (par. 3.5.) alla serie *denoised* (l'unica che verrà utilizzata di qui in avanti) sono rappresentati nella Figura

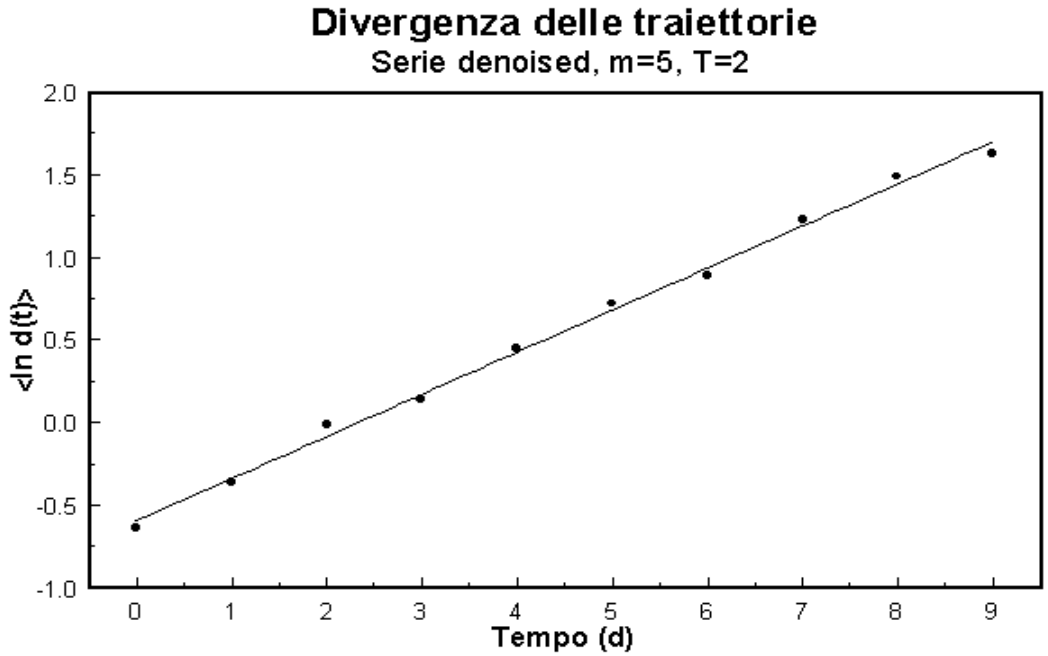

FIGURA 5.6 Divergenza media delle traiettorie per la serie delle temperature.

5.6. Il valore dell'esponente di Lyapunov maggiore risulta essere $\lambda_1 = 0.25$ il che conferma ulteriormente la caoticità della serie esaminata.

5.3 Un modello di previsione delle temperature

Dopo aver determinato il *time delay* e la dimensione di *embedding* è possibile ricostruire l'attrattore del sistema il quale, a sua volta, può essere usato per ottenere un modello della dinamica locale per predire l'evoluzione di un punto dello spazio delle fasi all'interno del bacino di attrazione nel quale tale punto si trova.

Il modello qui proposto si basa su di un'idea molto semplice: due punti vicini dell'attrattore tendono a separarsi a causa della divergenza delle traiettorie, ma per un tempo limitato (che dipende dall'esponente di Lyapunov) essi rimarranno abbastanza vicini in modo tale che l'evoluzione di uno segua quella dell'altro. Se dunque uno dei due punti è l'ultimo della serie, l'evoluzione temporale dell'altro può essere usata per ottenere una previsione dell'evoluzione del primo.

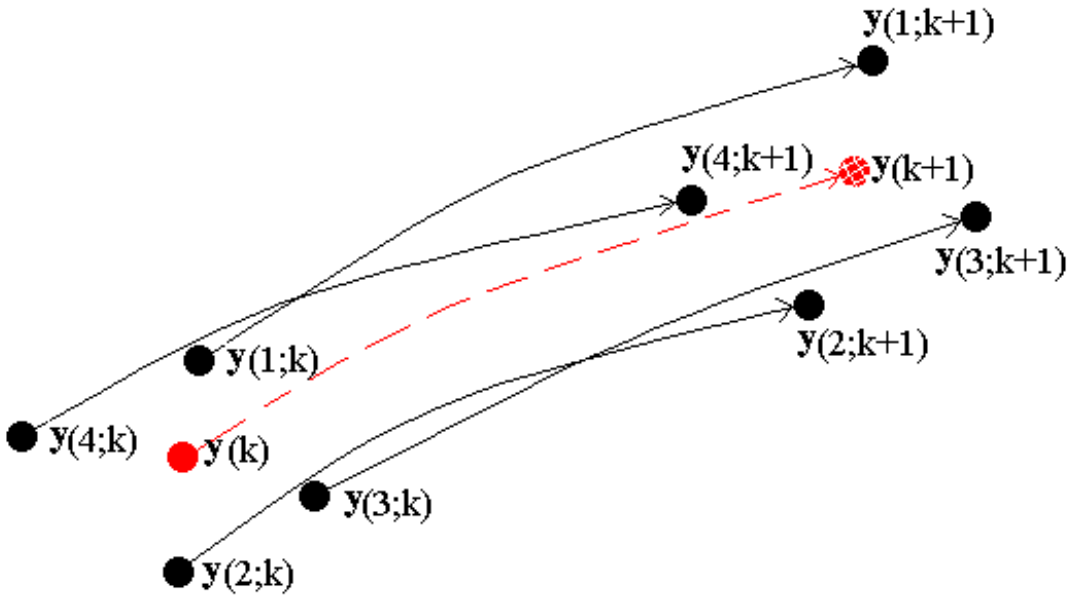

Figura 5.7 Rappresentazione schematica del modello previsionale: $\mathbf{y}(k)$ rappresenta il punto del quale si vuole prevedere l'evoluzione $\mathbf{y}(k+1)$, i punti neri rappresentano i primi vicini di $\mathbf{y}(k)$ con le rispettive evoluzioni.

La previsione può essere resa più accurata se, anziché considerare solo il primo vicino del punto del quale si vuole ottenere l'evoluzione, si calcola il punto previsto come una media dell'evoluzione di un certo numero N_B di punti vicini al punto dato. Nella Figura 5.7 è rappresentato schematicamente il modello.

Sia dunque $\mathbf{y}(k)$ il punto del quale si vuole predire l'evoluzione $\mathbf{y}(k+1)$; dati i primi N_B vicini di $\mathbf{y}(k)$: $\mathbf{y}(r;k)$; $r=1,2,\ldots,N_B$, si assume che $\mathbf{y}(k+1)$ sia dato dalla media pesata dei punti $\mathbf{y}(r;k+1)$, successivi dei punti $\mathbf{y}(r;k)$, con i pesi inversamente proporzionali alle distanze di tali punti da $\mathbf{y}(k)$:

$$\mathbf{y}(k+1)=\sum_{r=1}^{N_B} w_r \cdot \mathbf{y}(r;k+1) \qquad (5.1)$$

ove

$$w_r=\frac{d_r}{d}, \qquad d_r=\frac{1}{\left|\mathbf{y}(r;k)-\mathbf{y}(k)\right|}, \qquad d=\sum_{r=1}^{N_B}\frac{1}{d_r}.$$

Con una generalizzazione immediata il modello (5.1) può essere utilizzato per prevedere l'evoluzione di $\mathbf{y}(k)$ dopo *n* passi temporali:

$$\mathbf{y}(k+n)=\sum_{r=1}^{N_B} w_r \cdot \mathbf{y}(r;k+n). \qquad (5.2)$$

Ovviamente la qualità della previsione subirà un deterioramento all'aumentare di *n* a causa della divergenza delle traiettorie; infatti, se E_i è l'errore che si compie nella determinazione di $\mathbf{y}(k+i)$, in condizioni ideali l'errore nella previsione del passo temporale successivo sarà dato da $E_{i+1}=e^{\tau\lambda}E_i$ ove τ è il periodo di campionamento e λ il primo esponente di Lyapunov del sistema; essendo quest'ultimo positivo, l'errore tende necessariamente ad aumentare. Di fatto poiché il modello, approssimando un punto con una media pesata di punti vicini, inevitabilmente introduce degli errori indipendenti dalla divergenza delle traiettorie, la qualità della previsione sarà sempre peggiore di quella ideale.

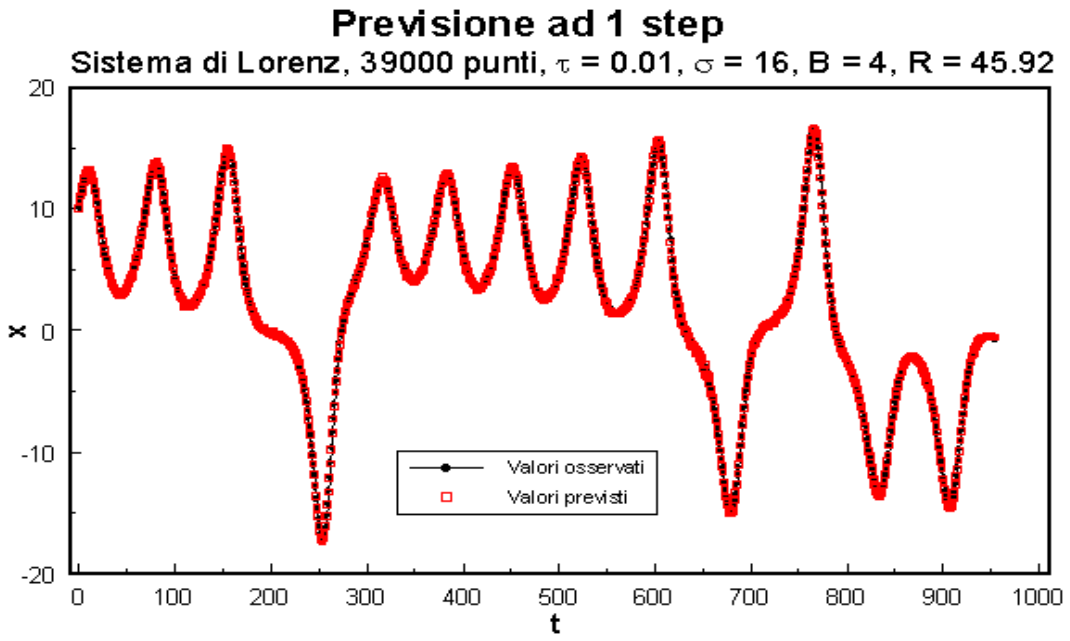

FIGURA 5.8 Confronto dei valori osservati con quelli previsti ad uno step d'integrazione per la variabile *x* del sistema di Lorenz.

Nelle Figure 5.8 e 5.9 sono rappresentate le previsioni, rispettivamente ad uno e tre passi temporali, relative alla variabile *x* del sistema di Lorenz:l'errore medio compiuto sulla previsione ad un passo è $E_1 = 1.03$, quello sulla previsione a

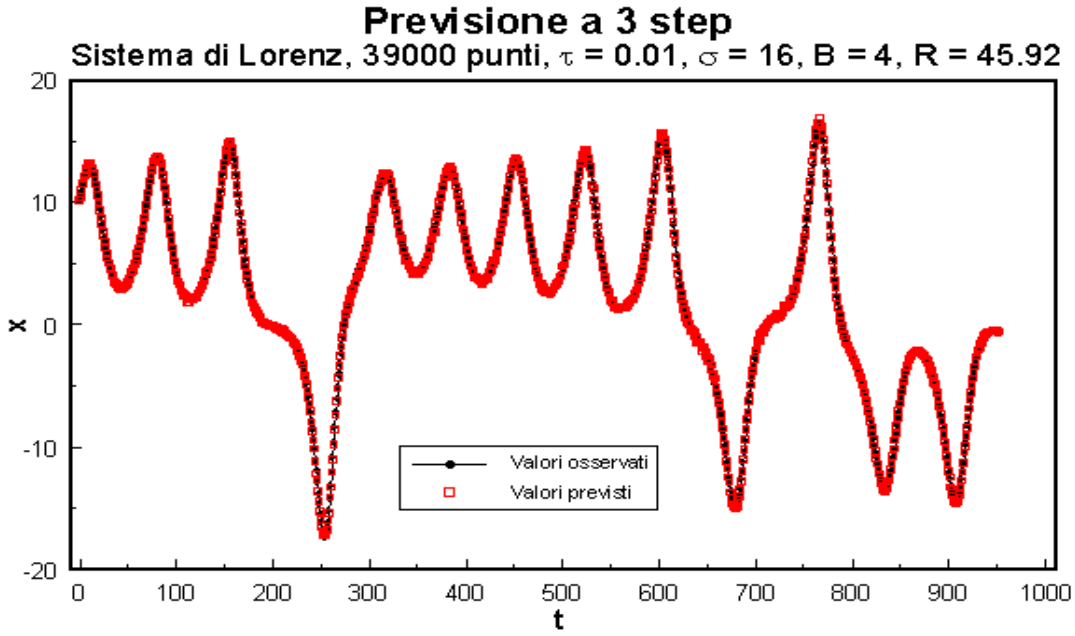

FIGURA 5.9 Confronto dei valori osservati con quelli previsti a tre step d'integrazione per la variabile *x* del sistema di Lorenz.

tre passi è $E_3 = 3.07$; se si confronta quest'ultimo con l'errore indotto dalla divergenza delle traiettorie: $E_3 = e^{2\tau\lambda} E_1 = 1.06$, si vede come l'errore indotto dal modello sia preponderante.

Nelle Figure 5.10 e 5.11 sono rappresentate le previsioni ad uno, due e tre giorni della serie delle temperature. Come si vede, nonostante gli errori introdotti, il modello consente di ottenere previsioni abbastanza buone.

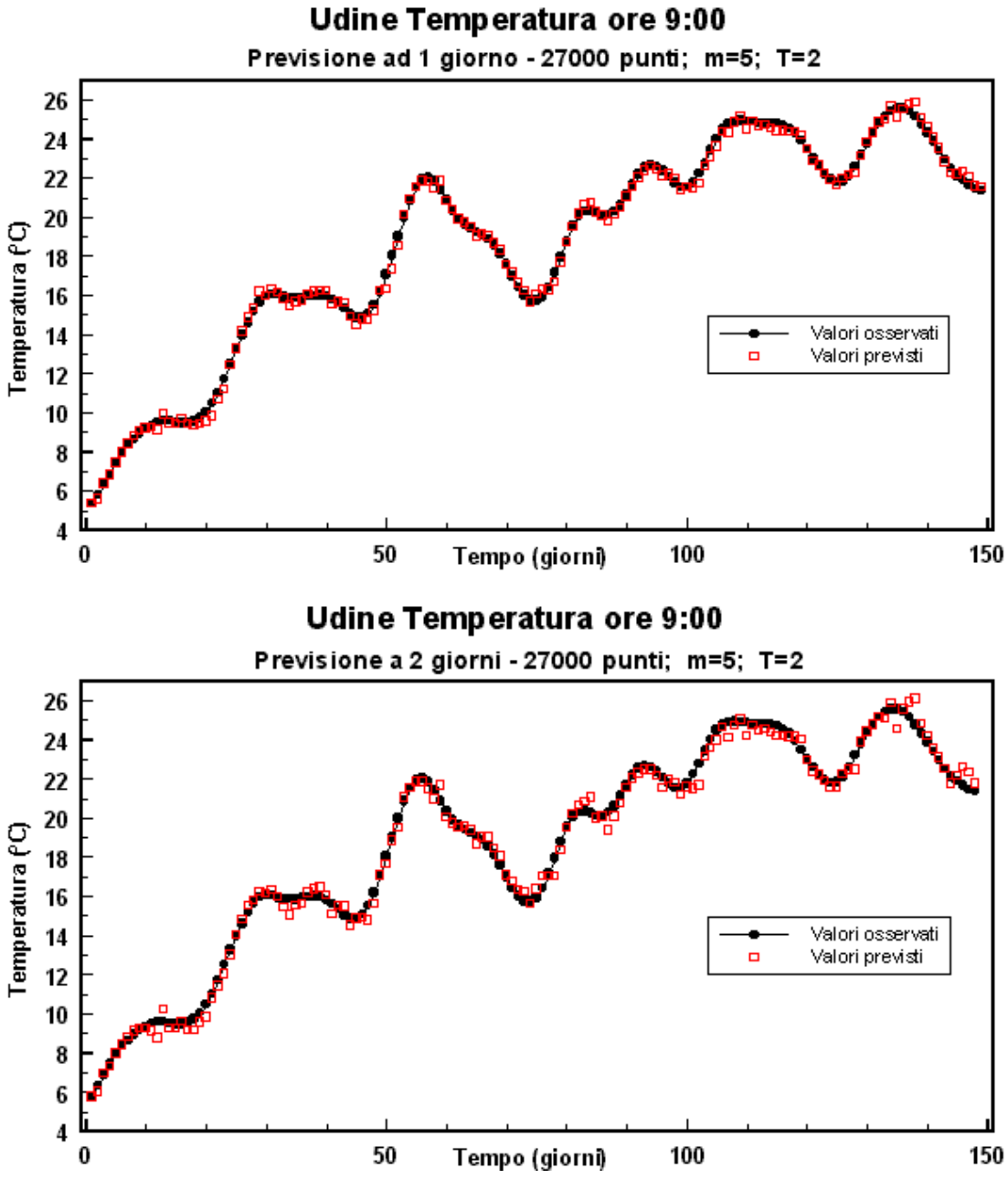

FIGURA 5.10 Confronto dei valori osservati con le previsioni ad uno e due giorni per la serie di temperature di Udine.

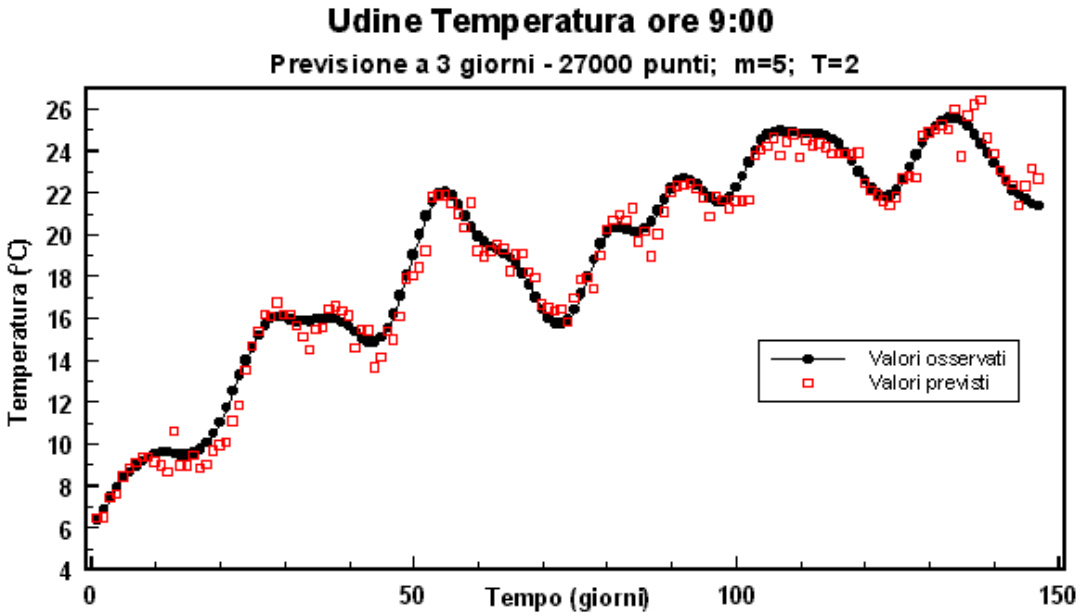

FIGURA 5.11 Confronto dei valori osservati con le previsioni a tre giorni per la serie di temperature di Udine.

Nella Tabella 5.1 sono riportati gli errori minimi, massimi e medi compiuti nelle previsioni ad uno, due e tre giorni delle Figure 5.10 e 5.9:

Ampiezza previsione (giorni)	Errore minimo (°C)	Errore massimo (°C)	Errore medio (°C)
1	0.00	0.75	0.20
2	0.01	1.32	0.33
3	0.02	2.06	0.58

TABELLA 5.1 Errori compiuti nelle previsioni del modello.

5.4 Risultati e conclusioni

L'applicazione del metodo dei falsi vicini ha permesso di dimostrare che la temperatura dell'aria di Udine-Castello è governata, almeno localmente, da un sistema caotico a 5 gradi di libertà e non da un sistema stocastico. Questo risultato è

stato confermato dal calcolo dell'esponente di Lyapunov maggiore. L'attrattore di tale sistema caotico nello spazio delle fasi a 5 dimensioni è stato ricostruito utilizzando il teorema di Takens ed avendo determinato il valore del *time delay* con il metodo della mutua informazione media.

Le informazioni sull'evoluzione temporale di un punto nello spazio delle fasi contenute nell'attrattore di un sistema caotico sono state utilizzate per sviluppare un modello locale di previsione della serie studiata. Tale modello consente di ottenere previsioni abbastanza buone, almeno entro un limite pari a tre volte il periodo di campionamento della serie in esame.

Le tecniche qui descritte hanno carattere generale nel senso che possono essere applicate ad una serie temporale a priori qualsiasi; e, qualora questa risultasse generata da un sistema caotico a basso numero di gradi di libertà, il modello proposto nel presente lavoro potrà essere utilizzato per ottenere delle previsioni.

A. Codice dei principali programmi usati

Nella presente appendice è riportato il codice sorgente dei principali programmi utilizzati nel presente lavoro.

I programmi sono stati scritti in linguaggio “C++” e compilati su PC con compilatore “Borland C++ 5.01”. In alcuni casi si è fatto ricorso a routine in “C” di provenienza esterna (W. H. Press et al., 1992).

A.1 Mutua Informazione Media

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#define NR_END 1
#define FREE_ARG char*
void nrerror(char error_text[])
{
  fprintf(stderr,"Numerical Recipes run-time error...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n\n");
  exit(1);
}
float *vector(long nl, long nh)
{
  float *v;
  v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
  if (!v) nrerror("allocation failure in vector()");
  return v-nl+NR_END;
}
void free_vector(float *v, long nl)
```

```
{
  free((FREE_ARG) (v+nl-NR_END));
}
float **matrix(long nrl, long nrh, long ncl, long nch)
{
  long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
  float **m;
  m=(float **) malloc((size_t)((nrow+1)*sizeof(float*)));
  if (!m) nrerror("allocation failure 1 in matrix()");
  m += 1;
  m -= nrl;
  m[nrl]=(float *) malloc((size_t)((nrow*ncol+1)*sizeof(float)));
  if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
  m[nrl] += 1;
  m[nrl] -= ncl;
  for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;
  return m;
}
void free_matrix(float **m, long nrl, long ncl)
{
  free((char*) (m[nrl]+ncl-1));
  free((char*) (m+nrl-1));
}
void main()
{
  float *x, *y, *Px, *Py, **Pxy, *IT;
  float xmin=100000.0, xmax=-100000.0, ymin, ymax;
  float c, cc, d=0.1, r=0.025;//, Pxtot=0.0;
  int i, m, n, N, Nx, Ny, T, Tmax;
  bool found;
  FILE *f_in, *f_out;
  char file_in[128], file_out[128];
  printf("\n Dare nome file input -> ");
  scanf("%s", file_in);
  printf("\n Dare N -> ");
  scanf("%d", &N);
  printf("\n Dare Tmax -> ");
  scanf("%d", &Tmax);
  printf("\n Dare nome file output -> ");
  scanf("%s", file_out);
  f_out = fopen(file_out, "w");
  IT = vector(0,Tmax);
  for(i=0; i<=Tmax; i++) IT[i]=0.0;
  x = vector(1,N+Tmax);
  if((f_in = fopen(file_in, "r"))==NULL) {
    printf("\nError opening file \"%s\".\nExiting program...",
file_in);
    exit(0);
  }
  for(i=1; i<=N+Tmax; i++) fscanf(f_in, "%f", &x[i]);
  fclose(f_in);
  for(i=1; i<=N; i++) {
    if(x[i]<=xmin) xmin=x[i];
    if(x[i]>=xmax) xmax=x[i];
  }
  Nx=(int)ceil((xmax-xmin)/d);
  Px = vector(0,Nx);
  for(i=0; i<=Nx; i++) Px[i]=0.0;
  for(i=1; i<=N; i++) {
    found=false;
    for(m=0; m<=Nx; m++) {
      c = xmin+m*d;
```

```
    if( (x[i]>c-r) && (x[i]<=c+r) ) {
      Px[m]++;
      found=true;
    }
    if(found) break;
  }
}
for(i=0; i<=Nx; i++) Px[i]/=N;
for(T=0; T<=Tmax; T++) {
  y = vector(1,N);
  for(i=1; i<=N; i++) y[i]=x[i+T];
  ymin=100000.0;
  ymax=-100000.0;
  for(i=1; i<=N; i++) {
    if(y[i]<=ymin) ymin=y[i];
    if(y[i]>=ymax) ymax=y[i];
  }
  Ny=(int)ceil((ymax-ymin)/d);
  Py=vector(0,Ny);
  for(i=0; i<=Ny; i++) Py[i]=0.0;
  for(i=1; i<=N; i++) {
    found=false;
    for(m=0; m<=Ny; m++) {
      c = ymin+m*d;
      if( (y[i]>c-r) && (y[i]<c+r) ) {
        Py[m]++;
        found=true;
      }
      if(found) break;
    }
  }
  for(i=0; i<=Ny; i++) Py[i]/=N;
  Pxy=matrix(0,Nx,0,Ny);
  for(m=0; m<=Nx; m++) for(n=0; n<=Ny; n++) Pxy[m][n]=0.0;
  for(i=1; i<=N; i++) {
    found=false;
    for(m=0; m<=Nx; m++) {
      c = xmin+m*d;
      if( (x[i]>c-r) && (x[i]<c+r) ) {
        for(n=0; n<=Ny; n++) {
          cc = ymin+n*d;
          if( (y[i]>cc-r) && (y[i]<cc+r) ) {
            Pxy[m][n]++;
            found=true;
          }
          if(found) break;
        }
      }
      if(found) break;
    }
  }
  for(m=0; m<=Nx; m++) for(n=0; n<=Ny; n++) Pxy[m][n]/=N;
  for(m=0; m<=Nx; m++)
    for(n=0; n<=Ny; n++)
      if(Px[m]!=0.0 && Py[n]!=0.0 && Pxy[m][n]!=0.0)
        IT[T]+=Pxy[m][n]*(log(Pxy[m][n]/(Px[m]*Py[n]))/log(2.0));
  free_vector(y,1);
  free_vector(Py,0);
  free_matrix(Pxy,0,0);
  printf(" IT[%2d] = %f\n", T, IT[T]);
}
free_vector(x,1);
```

```
  free_vector(Px,0);
  for(i=0; i<=Tmax; i++) fprintf(f_out, "%d\t%f\n", i, IT[i]);
  fclose(f_out);
  free_vector(IT,0);
}
```

A.2 Falsi Vicini

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>
#define NR_END 1
#define FREE_ARG char*
static float sqrarg=0.0;
#define SQR(a) (sqrarg=(a), sqrarg*sqrarg)
void nrerror(char error_text[])
{
  fprintf(stderr,"Numerical Recipes run-time error...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n\n");
  exit(1);
}
int *ivector(long nl, long nh)
{
  int *v;
  v=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
  if (!v) nrerror("allocation failure in ivector()");
  return v-nl+NR_END;
}
void free_ivector(int *v, long nl)
{
  free((FREE_ARG) (v+nl-NR_END));
}
float *vector(long nl, long nh)
{
  float *v;
  v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
  if (!v) nrerror("allocation failure in vector()");
  return v-nl+NR_END;
}
void free_vector(float *v, long nl)
{
  free((FREE_ARG) (v+nl-NR_END));
}
float RMS(float *x, int n)
{
  float mean=0.0, rms=0.0;
  int i;
  for(i=1; i<=n; i++) mean += x[i];
  mean /= n;
  for(i=1; i<=n; i++) rms += fabs(x[i]-mean);
  rms /= n;
  return rms;
}
int Nea_nei(float *x, int np, int m, int T, int k, float* d)
{
  float dist, dist_min = 1000000.0;
  int i, i_min, j;
```

```
  for(i=1; i<=np; i++) {
    if(i!=k) {
      dist = 0.0;
      for(j=0; j<m; j++) dist += SQR(x[k+j*T]-x[i+j*T]);
      dist = sqrt(dist);
      if(dist<=dist_min) {
        dist_min = dist;
        i_min = i;
      }
    }
  }
  if(dist_min == 0.0) dist_min = 0.00000001;
  *d = dist_min;
  return i_min;
}
void main()
{
  int i, m, M_MAX, T;
  int N=0;
  float x, *y, rms;
  FILE *f_in, *f_out;
  char file_in[128], file_out[128];
  printf("\n Dare nome file input -> ");
  scanf("%s", file_in);
  if((f_in = fopen(file_in, "r"))==NULL) {
    printf("\nError opening file \"%s\".\nExiting program...\n\n",
file_in);
    exit(0);
  }
  while(!feof(f_in)) {
    fscanf(f_in, "%f", &x);
    N++;
  }
  printf("\n %d points read.\n", N);
  rewind(f_in);
  y = vector(1,N);
  for(i=1; i<=N; i++) fscanf(f_in, "%f", &y[i]);
  fclose(f_in);
  rms = RMS(y, N);
  printf("\n Dare nome file output -> ");
  scanf("%s", file_out);
  printf("\n Dare m max -> ");
  scanf("%d", &M_MAX);
  printf("\n Dare T -> ");
  scanf("%d", &T);
  printf("\n");
  f_out = fopen(file_out, "w");
  _setcursortype(_NOCURSOR);
  for(m=1; m<=M_MAX; m++) {
    int *nn, NP=N-(m-1)*T, fnn=0, px, py;
    float *dists, num1, num2, d;
    dists = vector(1,NP);
    nn = ivector(1,NP);
    printf(" m = %2d", m);
    px = wherex();
    py = wherey();
    for(i=1; i<=NP; i++) {
      nn[i] = Nea_nei(y, NP, m, T, i, &d);
      dists[i] = d;
      gotoxy(px+3, py);
      printf("%3d%%", i*100/NP);
    }
```

```
    for(i=1; i<=NP; i++) {
      num1 = fabs(y[i+m*T]-y[nn[i]+m*T]);
      num2 = sqrt(SQR(dists[i])+SQR(num1));
      if( ((num1/dists[i])>15.0) || ((num2/rms)>=2.0) ) fnn++;
    }
    free_vector(dists,1);
    free_ivector(nn,1);
    fprintf(f_out,"%d\t%f\t%d\t%d\n", m, (fnn*100.0)/NP, fnn, NP);
    printf("\n");
  }
  _setcursortype(_NORMALCURSOR);
  free_vector(y,1);
  fclose(f_out);
}
```

A.3 Box Counting Dimension

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#define LN2 0.69314718
#define MIN(A,B) ((A)<(B))?(A):(B)
#define NR_END 1
#define FREE_ARG char*
void nrerror(char error_text[])
{
  fprintf(stderr,"Numerical Recipes run-time error...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n\n");
  exit(1);
}
float *vector(long nl, long nh)
{
  float *v;
  v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
  if (!v) nrerror("allocation failure in vector()");
  return v-nl+NR_END;
}
void free_vector(float *v, long nl)
{
  free((FREE_ARG) (v+nl-NR_END));
}
int *ivector(long nl, long nh)
{
  int *w;
  w=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
  if (!w) nrerror("allocation failure in vector()");
  return w-nl+NR_END;
}
void free_ivector(int *w, long nl)
{
  free((FREE_ARG) (w+nl-NR_END));
}
void main()
{
  float *x, *y, xmin, ymin, a, b, d, x0, y0, w, h;
  int *box, i, j, k, N=0, Db, ND, ndx, ndy;
  bool found;
  FILE *f_in;
```

```
  if((f_in = fopen("henon.txt", "r"))==NULL) {
    printf("\nError opening file \"%s\".\nExiting program...",
"henon.txt");
    exit(0);
  }
  while(!feof(f_in)) {
    fscanf(f_in, "%f\t%f", &a, &b);
    N++;
  }
  rewind(f_in);
  x=vector(1,N);
  y=vector(1,N);
  for(i=1; i<=N; i++) fscanf(f_in, "%f\t%f", &x[i], &y[i]);
  fclose(f_in);
  printf("\n %d points read.\n", N);
  printf("\n");
  x0=-1.5;
  y0=-0.4;
  w=3.0;
  h=0.8;
  for(i=9; i<=10; i++) {
    Db=0;
    d=MIN(w,h)/pow(2,i);
    ndx=(int)ceil(w/d);
    ndy=(int)ceil(h/d);
    ND=ndx*ndy;
    box=ivector(0,ND-1);
    for(j=0; j<ND; j++) box[j]=0;
    printf("\n ND=%8d\t", ND);
    for(k=0; k<ND;k++) {
      found=false;
      xmin=x0+(k%ndx)*d;
      ymin=y0+(k/ndx)*d;
      for(j=1; j<=N; j++) {
        if( (x[j]>xmin) && (x[j]<=xmin+d) &&
           (y[j]>ymin) && (y[j]<=ymin+d) ) found=true;
        if(found) break;
      }
      if(found) box[k]=1;
    }
    for(j=0; j<ND; j++) Db+=box[j];
    printf("log2(1/d)=%f\tlog2(N(d))=%f", log(1.0/d)/LN2,
log(Db)/LN2);
    free_ivector(box,0);
  }
  free_vector(x,1);
  free_vector(y,1);
}
```

A.4 Dimensione di Informazione

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#define LN2 0.69314718
#define MIN(A,B) ((A)<(B))?(A):(B)
#define NR_END 1
#define FREE_ARG char*
void nrerror(char error_text[])
```

```
{
  fprintf(stderr,"Numerical Recipes run-time error...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n\n");
  exit(1);
}
float *vector(long nl, long nh)
{
  float *v;
  v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
  if (!v) nrerror("allocation failure in vector()");
  return v-nl+NR_END;
}
void free_vector(float *v, long nl)
{
  free((FREE_ARG) (v+nl-NR_END));
}
int *ivector(long nl, long nh)
{
  int *w;
  w=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
  if (!w) nrerror("allocation failure in vector()");
  return w-nl+NR_END;
}
void free_ivector(int *w, long nl)
{
  free((FREE_ARG) (w+nl-NR_END));
}
void main()
{
  float *x, *y, *box, xmin, ymin, a, b, d, x0, y0, w, h, Is;
  int i, j, k, N=0, ND, ndx, ndy;
  FILE *f_in;
  if((f_in = fopen("henon.txt", "r"))==NULL) {
    printf("\nError opening file \"%s\".\nExiting program...",
"henon.txt");
    exit(0);
  }
  while(!feof(f_in)) {
    fscanf(f_in, "%f\t%f", &a, &b);
    N++;
  }
  rewind(f_in);
  x=vector(1,N);
  y=vector(1,N);
  for(i=1; i<=N; i++) fscanf(f_in, "%f\t%f", &x[i], &y[i]);
  fclose(f_in);
  printf("\n %d points read.\n", N);
  x0=-1.5;
  y0=-0.4;
  w=3.0;
  h=0.8;
  for(i=2; i<=10; i++) {
    Is=0.0;
    d=MIN(w,h)/pow(2,i);
    ndx=(int)ceil(w/d);
    ndy=(int)ceil(h/d);
    ND=ndx*ndy;
    box=vector(0,ND-1);
    for(j=0; j<ND; j++) box[j]=0;
    printf("\n ND=%8d\t", ND);
    for(k=0; k<ND;k++) {
```

```
      xmin=x0+(k%ndx)*d;
      ymin=y0+(k/ndx)*d;
      for(j=1; j<=N; j++) {
        if( (x[j]>xmin) && (x[j]<=xmin+d) &&
            (y[j]>ymin) && (y[j]<=ymin+d) ) box[k]++;
      }
    }
    for(j=0; j<ND; j++)  box[j]/=N;
    for(j=0; j<ND; j++)
      if(box[j]>0.0)
        Is+=box[j]*log(1.0/box[j])/LN2;
    printf("log2(1/d)=%f\tI(d)=%f", log(1.0/d)/LN2, Is);
    free_vector(box,0);
  }
  free_vector(x,1);
  free_vector(y,1);
}
```

A.5 Funzione di Correlazione

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>
#define SWAP2(a,b) temp=(a);(a)=(b);(b)=temp;
#define M2 3                /* serve a sort */
#define NSTACK 50           /* serve a sort */
#define MAX(A,B) ((A)>(B))?(A):(B)
#define N_R  1000
#define M_MIN  1
#define M_MAX  10
#define T 1
#define N_ROW   (M_MAX - M_MIN + 1)*3
#define BUF 150
#define NR_END 1
#define FREE_ARG char*
void nrerror(char error_text[])
{
  fprintf(stderr,"Numerical Recipes run-time error...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n\n");
  exit(1);
}
float *vector(long nl, long nh)
{
  float *v;
  v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
  if (!v) nrerror("allocation failure in vector()");
  return v-nl+NR_END;
}
int *ivector(long nl, long nh)
{
  int *w;
  w=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
  if (!w) nrerror("allocation failure in vector()");
  return w-nl+NR_END;
}
void free_vector(float *v, long nl)
{
```

```
  free((FREE_ARG) (v+nl-NR_END));
}
void free_ivector(int *w, long nl)
{
  free((FREE_ARG) (w+nl-NR_END));
}
float Alpha(float *x, int n);
void Derive(float y[], float h, int n);
void sort(unsigned long n, float arr[]);
void main()
{
  int Np=0, i, j, k, m, W;
  float out[N_ROW][N_R], c[N_R], x, *y, r_max=1.0;
  FILE *f_in, *f_out;
  char file_in[BUF];//, file_out[BUF];
  printf("\n Dare nome file input -> ");
  scanf("%s", file_in);
  if((f_in = fopen(file_in, "r"))==NULL) {
    printf("\nError opening file \"%s\".\nExiting program...",
file_in);
    exit(0);
  }
  while(!feof(f_in)) {
    fscanf(f_in, "%f", &x);
    Np++;
  }
  printf("\n %d points read.\n", Np);
  rewind(f_in);
  y=vector(1,Np);
  for(i=1; i<=Np; i++) fscanf(f_in, "%f", &y[i]);
  fclose(f_in);
  W = (int)(1.0/log(1.0/Alpha(y, Np)));
  printf("\n\n Calculating distances... ");
  for(m=M_MIN; m<=M_MAX; m++)
  {
    int N=Np-(m-1)*T;
    long ND0=(N*(N-W-2))/2, ND;
    long i_d=1, j_d=1;
    float *d, a, b, h, r_tmp;
    printf("\n ND = %d", ND0);
    printf("\tm = %2d... ", m);
    d=vector(1,ND0);
    for(j=1; j<N-W; j++) {
      for(i=j+W+1; i<=N; i++) {
        float d_elem = 0.0;
        for(k=0; k<m; k++)
          d_elem += (y[j+k*T]-y[i+k*T])*(y[j+k*T]-y[i+k*T]);
        d_elem = sqrt(d_elem);
        if((d_elem>0) && (d_elem<r_max)) d[i_d++] = d_elem;
      }
    }
    printf("done.");
    ND=i_d;
    printf("  Sorting %d numbers... ", ND);
    sort(ND, d);
    printf("done.");
    a=log10(d[10]);
    b=log10(r_max);
    h=(b-a)/(N_R-1);
    for(i=0; i<N_R; i++)
    {
      r_tmp = pow(10.0, a+i*h);
```

```
      out[(m-M_MIN)*3][i]=log10(r_tmp);
      while((d[j_d]<r_tmp) && (j_d<=ND)) j_d++;
      c[i] = log10((float)j_d/(float)ND);
      out[(m-M_MIN)*3+1][i]=c[i];
    }
    free_vector(d,1);
    Derive(c, h, N_R);
    for(i=0; i<N_R; i++) out[(m-M_MIN)*3+2][i]=c[i];
  }
  free_vector(y,1);
  printf("\n done.");
  f_out = fopen("Udine_gpa.txt"/*file_out*/, "w");
  printf("\n\n Writing output... ");
  for(i=0; i<N_ROW/3; i++) fprintf(f_out, "r%d\tc%d\tdc%d\t",
                        i+M_MIN, i+M_MIN, i+M_MIN);
  fprintf(f_out, "\n");
  for(i=0; i<N_R; i++)
  {
    for(j=0; j<N_ROW; j++) fprintf(f_out, "%f\t", out[j][i]);
    fprintf(f_out, "\n");
  }
  fclose(f_out);
  printf("done.\n\n\tProgram Terminated.\n");
}
float Alpha(float *x, int n)
{
  float mean=0.0, sigma2=0.0, alpha=0.0;
  int i;
  for(i=1; i<=n; i++) mean += x[i];
  mean /= n;
  for(i=1; i<=n; i++) sigma2 += (x[i]-mean)*(x[i]-mean);
  for(i=1; i<=n; i++) alpha += (x[i+1]-mean)*(x[i]-mean);
  alpha /= sigma2;
  return alpha;
}
void Derive(float y[], float h, int n)
{
  float *t;
  float a0=1.0/12.0, a2=25.0/12.0, a4=4.0/3.0, a6=5.0/6.0,
a8=2.0/3.0;
  float a1=-1.0*a0, a3=-1.0*a2, a5=-1.0*a4, a7=-1.0*a6, a9=-1.0*a8;
  int i;
  if((t = calloc(n, sizeof(float))) == NULL)
  {
    printf("\nNot enough memory to allocate array\nExiting
program...");
    exit(0);
  }
  for(i=0; i<n; i++) t[i] = y[i];
  y[0] = (a3*t[0]+4*t[1]-3*t[2]+a4*t[3]-0.25*t[4])/h;
  y[1] = (0.25*t[0]+a7*t[1]+1.5*t[2]-0.5*t[3]+a0*t[4])/h;
  for(i=2; i<n-2; i++) y[i] = (a0*t[i-2]+a9*t[i-
1]+a8*t[i+1]+a1*t[i+2])/h;
  y[n-2] = (a1*t[n-5]+0.5*t[n-4]-1.5*t[n-3]+a6*t[n-2]+0.25*t[n-
1])/h;
  y[n-1] = (0.25*t[n-5]+a5*t[n-4]+3*t[n-3]-4*t[n-2]+a2*t[n-1])/h;
  free(t);
}
void sort(unsigned long n, float arr[])
{
  unsigned long i, ir=n, j, k, l=1;
  int jstack=0, *istack;
```

```
  float a, temp;
  istack = ivector(0,NSTACK);
  for(;;) {
    if(ir-l < M2) {
      for(j=l+1; j<=ir; j++) {
        a=arr[j];
        for(i=j-1; i>=1; i--) {
          if(arr[i] <= a ) break;
          arr[i+1]=arr[i];
        }
        arr[i+1] = a;
      }
      if(jstack == 0) break;
      ir=istack[jstack];
      l =istack[jstack-1];
      jstack-=2;
    }
    else {
      k = (l+ir) >> 1;
      SWAP2(arr[k], arr[l+1]);
      if(arr[l+1] > arr[ir])
      {
        SWAP2(arr[l+1], arr[ir]);
      }
      if(arr[l] > arr[ir])
      {
        SWAP2(arr[l], arr[ir]);
      }
      if(arr[l+1] > arr[l])
      {
        SWAP2(arr[l+1], arr[l]);
      }
      i=l+1;
      j=ir;
      a=arr[l];
      for(;;) {
        do i++; while(arr[i] < a);
        do j--; while(arr[j] > a);
        if(j<i) break;
        SWAP2(arr[i], arr[j]);
      }
      arr[l]=arr[j];
      arr[j]=a;
      jstack += 2;
      if(jstack > NSTACK) {
        printf("jstack > NSTACK");
        exit(0);
      }
      if(ir-i+1 >= j-l) {
        istack[jstack]=ir;
        istack[jstack-1]=i;
        ir=j-1;
      }
      else {
        istack[jstack]=j-1;
        istack[jstack-1]=l;
        l=i;
      }
    }
  }
  free_ivector(istack,0);
}
```

A.6 Esponente di Lyapunov

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>
#define NR_END 1
#define FREE_ARG char*
static float sqrarg=0.0;
#define SQR(a) (sqrarg=(a), sqrarg*sqrarg)
void nrerror(char error_text[])
{
  fprintf(stderr,"Numerical Recipes run-time error...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n\n");
  exit(1);
}
int *ivector(long nl, long nh)
{
  int *v;
  v=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
  if (!v) nrerror("allocation failure in ivector()");
  return v-nl+NR_END;
}
void free_ivector(int *v, long nl)
{
  free((FREE_ARG) (v+nl-NR_END));
}
float *vector(long nl, long nh)
{
  float *v;
  v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
  if (!v) nrerror("allocation failure in vector()");
  return v-nl+NR_END;
}
void free_vector(float *v, long nl)
{
  free((FREE_ARG) (v+nl-NR_END));
}
int Nea_nei(float *x, int np, int m, int T, int k)
{
  float dist, dist_min = 1000000.0;
  int i, i_min, j;
  for(i=1; i<=np; i++) {
    if(abs(i-k)>0) {      //periodo medio
      dist=0.0;
      for(j=0; j<m; j++) dist+=SQR(x[k+j*T]-x[i+j*T]);
      dist = sqrt(dist);
      if(dist<=dist_min) {
        dist_min=dist;
        i_min = i;
      }
    }
  }
  return i_min;
}
void main()
{
  int ND=0, NP, M, T, P, Ndist;
  int *nn;
  float dist, e, x, dt;
```

```
  float *y;
  FILE *f_in, *f_out;
  char file_in[128], file_out[128];
  printf("\n Nome file input -> ");
  scanf("%s", file_in);
  if((f_in = fopen(file_in, "r"))==NULL) {
    printf("\nError opening file \"%s\".\nExiting program...\n\n",
file_in);
    exit(0);
  }
  while(!feof(f_in)) {
    fscanf(f_in, "%f", &x);    //conta i dati
    ND++;
  }
  printf("\n Letti %d dati.\n", ND);
  rewind(f_in);
  y = vector(1,ND);
  for(int i=1; i<=ND; i++) fscanf(f_in, "%f", &y[i]); //legge i dati
  fclose(f_in);
  printf("\n Dimensione di embedding -> ");
  scanf("%d", &M);
  printf("\n Time delay -> ");
  scanf("%d", &T);
  printf("\n Tempo di campionamento -> ");
  scanf("%f", &dt);
  NP=ND-(M-1)*T;      //calcola il numero di punti dell'orbita
  printf("\n Step max -> ");
  scanf("%d", &P);
  nn=ivector(1,NP);
  printf("\n Nome file output -> ");
  scanf("%s", file_out);
  printf("\n");
  f_out = fopen(file_out, "w");
  for(int i=1; i<=NP; i++) nn[i]=Nea_nei(y, NP, M, T, i);
  for(int i=0; i<=P; i++) {
    Ndist=0;
    e=0;
    for(int j=1; j<=NP; j++) {
      dist=0.0;
      if(((j+i)<=NP) && ((nn[j]+i)<=NP)) {
        for(int k=0; k<M; k++)
          dist+=SQR(y[j+k*T+i]-y[nn[j]+k*T+i]);
      }
      if(dist>0) {
        Ndist++;
        e+=log(sqrt(dist));
      }
    }
    e/=Ndist;
    fprintf(f_out,"%f\t%f\n", i*dt, e);
  }
  free_vector(y,1);
  free_ivector(nn,1);
  fclose(f_out);
}
```

A.7 Wavelet Transform

```
#include <stdlib.h>
```

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <fstream.h>
#include <classlib\arrays.h>
static float sqrarg;
#define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)
inline float Arg(int x, int b, int a)
{
  return SQR(((float)x-b)/a);
}
inline float Psi(int x, int b, int a)
{
  return (1-Arg(x,b,a))*exp(-Arg(x,b,a)/2.0);
}
typedef TArray<float> Vector;
typedef TArrayIterator<float> VectorIter;
class Line : public Vector
{
  public :
    Line() : Vector(1000, 1, 100) {}
    bool operator ==(const Line& other) const
      { return &other == this; }
};
typedef TArray<Line> Matrix;
typedef TArrayIterator<Line> MatrixIter;
char Question(char* qst);
void WriteFile(char* filename, Vector x);
class TimeSerie
{
  public :
    Vector* data;
    int ndata;
    TimeSerie(char* filein);
    ~TimeSerie() { delete data; }
    Matrix Analysis(int amax);
    Vector GetData();
    void WriteOutput(char* fileout, Matrix coeff, int amax);
};
class WaveCoeff
{
  public:
    Matrix* coeff;
    int na, nb;
    WaveCoeff(char* filein);
    WaveCoeff(Matrix data_in, int na_in, int nb_in);
    ~WaveCoeff() { delete coeff; }
    Vector Synthesis(int rec_amin, int rec_amax);
};
TimeSerie::TimeSerie(char* filein)
{
  data = new Vector(1000, 1, 100);
  ifstream is(filein);
  if (!is)
  {
    printf("Can't open file %s", filein);
    exit(0);
  }
  data->Flush();
  while( !is.eof() )
  {
    float in_elem;
```

```
    is >> in_elem;
    data->Add(in_elem);
  }
  int n = data->GetItemsInContainer();
  data->Detach(n-1);
  ndata = data->GetItemsInContainer();
  printf("\n  ndata = %d", ndata);
}
Matrix
TimeSerie::Analysis(int amax)
{
  int yp = wherey()+1;
  gotoxy(3, yp);
  printf("Calculating transform coefficients [");
  int xp = wherex();
  Matrix coeff(amax, 1, 0);
  Line line;
  float out_elem;
  int xmin, xmax;
  for(int a=1; a<=amax; a++)
  {
    line.Flush();
    for(int b=1; b<=ndata; b++)
    {
      xmin = b - 4*a;
      xmax = b + 4*a;
      out_elem = 0.0;
      for(int x=xmin; x<=xmax; x++)
      {
        int i = x;
        if(    x<1) i = 1-x;
        if(x>ndata) i = 2*ndata-x+1;
        out_elem += Psi(x,b,a)*(*data)[i];
      }
      out_elem /= sqrt(a);
      line.Add(out_elem);
      gotoxy(xp, yp);
      printf("%3d,%4d]", a, b);
    }
    coeff.Add(line);
  }
  return coeff;
}
Vector
TimeSerie::GetData()
{
  Vector out(1000, 1, 100);
  VectorIter vi(*data);
  while(vi) out.Add(vi++);
  return out;
}
void
TimeSerie::WriteOutput(char* fileout, Matrix coeff, int amax)
{
  ofstream os(fileout);
  if(!os)
  {
    printf("Can't open file.");
    exit(0);
  }
  os <<'#'<<"\t"<<amax<<"\t"<<ndata<<"\n";
  MatrixIter ma(coeff);
```

```
  while(ma)
  {
    VectorIter ve(ma++);
    while( ve ) os << ve++ << "\t";
    os << "\n";
  }
  printf("\n\n  Coefficients written on file: %s", fileout);
}
WaveCoeff::WaveCoeff(char* filein)
{
  coeff = new Matrix(30, 1, 10);
  Line line;
  char dummy;
  ifstream is(filein);
  if (!is)
  {
    printf("Can't open file.");
    exit(0);
  }
  printf("\n  Reading data... ");
  is >> dummy >> na >> nb;
  coeff->Flush();
  float elem;
  for(int i=1; i<=na; i++)
  {
    line.Flush();
    for(int j=1; j<=nb; j++)
    {
      is >> elem;
      line.Add(elem);
    }
    coeff->Add(line);
  }
  printf("done.");
}
WaveCoeff::WaveCoeff(Matrix data_in, int na_in, int nb_in)
{
  coeff = new Matrix(30, 1, 10);
  Line line;
  MatrixIter mi(data_in);
  while(mi)
  {
    line.Flush();
    VectorIter vi(mi++);
    while( vi ) line.Add(vi++);
    coeff->Add(line);
  }
  na = na_in;
  nb = nb_in;
}
Vector
WaveCoeff::Synthesis(int rec_amin, int rec_amax)
{
  int yp = wherey()+1;
  gotoxy(3, yp);
  printf("Calculating reconstructed vector [");
  int xp = wherex();
  Vector out_vec(1000, 1, 100);
  Line line;
  float tmp_elem, out_elem;
  int bmin, bmax;
  out_vec.Flush();
```

```
  for(int x=1; x<=nb; x++)
  {
    out_elem = 0;
    MatrixIter mi(*coeff);
    mi.Restart(rec_amin, rec_amax);
    int a = rec_amin;
    while(mi)
    {
      tmp_elem = 0;
      bmin = x - 4*a;
      bmax = x + 4*a;
      for(int b=bmin; b<=bmax; b++)
      {
        int i = b;
        if( b<1) i = 1-b;
        if(b>nb) i = 2*nb-b+1;
        tmp_elem += Psi(x,b,a)*(mi.Current())[i];
      }
      out_elem += tmp_elem/pow(a,2.5);
      a++;
      mi++;
    }
    out_vec.Add(out_elem/3.14);
    gotoxy(xp, yp);
    printf("%d]", x);
  }
  return out_vec;
}
void main()
{
  int rec_amin_in, rec_amax_in;
  char answer = Question("Do you want to perform data analysis
(y/n)?");
  TimeSerie tmserie("input01.txt");
  Vector signal = tmserie.GetData();
  if( answer == 'y')
  {
    int amax_in;
    printf("\n  Input a max -> ");
    scanf("%d", &amax_in);
    Matrix coeff_in = tmserie.Analysis(amax_in);
    tmserie.WriteOutput("coeff.txt", coeff_in, amax_in);
    WaveCoeff wcoeff(coeff_in, amax_in, tmserie.ndata);
    printf("\n\n  Input a rec min ( >= 1 ) -> ");
    scanf("%d", &rec_amin_in);
    printf("\n  Input a rec max ( <= %d ) -> ", amax_in);
    scanf("%d", &rec_amax_in);
    Vector recostr = wcoeff.Synthesis(rec_amin_in, rec_amax_in);
    WriteFile("nn_rec.txt", recostr);
  }
  else
  {
    WaveCoeff wcoeff("coeff.txt");
    printf("\n\n  Input a rec min ( >= 1 ) -> ");
    scanf("%d", &rec_amin_in);
    printf("\n  Input a rec max ( <= %d ) -> ", wcoeff.na);
    scanf("%d", &rec_amax_in);
    Vector recostr = wcoeff.Synthesis(rec_amin_in, rec_amax_in);
    WriteFile("nn_rec.txt", recostr);
  }
  printf("\n\n  End of Program.");
}
```

```
char Question(char* qst)
{
  char answ = 'x';
  while( (answ != 'y') && (answ != 'n') )
  {
    printf("  %s ", qst);
    scanf("%c", &answ);
    if( answ == 'Y' ) answ = 'y';
    if( answ == 'N' ) answ = 'n';
  }
  return answ;
}
void WriteFile(char* filename, Vector x)
{
  ofstream os(filename);
  if(!os)
  {
    printf("Can't open file %s aborting.", filename);
    exit(0);
  }
  VectorIter xi(x);
  while( xi ) os << xi++  << "\n";
  printf("\n\n  Not normalized vector written on file: %s",
filename);
}
```

A.8 Modello Previsionale

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#define NR_END 1
#define FREE_ARG char*
static float sqrarg;
#define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)
void nrerror(char error_text[])
{
  fprintf(stderr,"Numerical Recipes run-time error...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n\n");
  exit(1);
}
int *ivector(long nl, long nh)
{
  int *v;
  v=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
  if (!v) nrerror("allocation failure in ivector()");
  return v-nl+NR_END;
}
void free_ivector(int *v, long nl)
{
  free((FREE_ARG) (v+nl-NR_END));
}
float *vector(long nl, long nh)
{
  float *v;
  v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
  if (!v) nrerror("allocation failure in vector()");
  return v-nl+NR_END;
```

```
}
void free_vector(float *v, long nl)
{
  free((FREE_ARG) (v+nl-NR_END));
}
float **matrix(long nrl, long nrh, long ncl, long nch)
{
  long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
  float **m;
  m=(float **) malloc((size_t)((nrow+1)*sizeof(float*)));
  if (!m) nrerror("allocation failure 1 in matrix()");
  m += 1;
  m -= nrl;
  m[nrl]=(float *) malloc((size_t)((nrow*ncol+1)*sizeof(float)));
  if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
  m[nrl] += 1;
  m[nrl] -= ncl;
  for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;
  return m;
}

void free_matrix(float **m, long nrl, long ncl)
{
  free((char*) (m[nrl]+ncl-1));
  free((char*) (m+nrl-1));
}
bool Check(int i, int *y, int n)
{
  for(int j=1; j<=n; j++) if(y[j]==i) return false;
  return true;
}
int PrimoVicino(float **yo, float **yl, int i, int nr, int nc)
{
  int j, k, index=0;
  float d, d_min=10000000.;
  for(j=1; j<=nr; j++) {
    d=0.0;
    for(k=1; k<=nc; k++) d+=SQR(yo[i][k]-yl[j][k]);
    d=sqrt(d);
    if(d<d_min) {
      d_min=d;
      index=j;
    }
  }
  return index;
}
void PrimiVicini(float **yl, int *y_idx, int nn, int nr, int nc, int
Nx)
{
  int i, j, k, index=0;
  float d, d_min=10000000.;
  for(i=1; i<=Nx; i++) y_idx[i]=nn;
  for(i=2; i<=Nx; i++) {
    for(j=1; j<=nr; j++) {
      if( Check(j,y_idx,Nx) ) {
        d=0.0;
        for(k=1; k<=nc; k++) d+=SQR(yl[nn][k]-yl[j][k]);
        d=sqrt(d);
        if(d<d_min) {
          d_min=d;
          index=j;
        }
```

```
      }
    }
    y_idx[i]=index;
  }
}
float Dist(float **a, float **b, int ia, int ib, int m)
{
  float d = 0.0;
  for(int j=1; j<=m; j++) d+=SQR(a[ia][j]-b[ib][j]);
  if(d>0) return sqrt(d);
  else return 0.000001;
}
void Weight(float *dst, int n)
{
  float d_tot=0.0;
  for(int i=1; i<=n; i++)
  {
    dst[i] = 1.0/dst[i];
    d_tot += dst[i];
  }
  for(int i=1; i<=n; i++) dst[i] /= d_tot;
}
void main()
{
  int Ntot=0, N, NL, NP, NO, M, T, Nx, Ndays;
  int *y_idx;
  float x, *y, *dists, **yl, **yo, **yp, eqm=0.0;
  FILE *f_in, *f_out;
  char file_in[128], file_out[128];
  printf("\n Nome file input -> ");
  scanf("%s", file_in);
  if((f_in = fopen(file_in, "r"))==NULL) {
    printf("\n Error opening file \"%s\".\n Exiting program...",
file_in);
    exit(0);
  }
  while(!feof(f_in)) {
    fscanf(f_in, "%f", &x);
    Ntot++;
  }
  rewind(f_in);
  printf("\n Letti %d dati dal file \"%s\".\n", Ntot, file_in);
  printf("\n Dimensione di embedding -> ");
  scanf("%d", &M);
  printf("\n Delay time -> ");
  scanf("%d", &T);
  N=Ntot-(M-1)*T;
  printf("\n Numero di punti sull'attrattore: %d.\n", N);
  printf("\n Learning points (NL) -> ");
  scanf("%d", &NL);
  printf("\n Ampiezza previsione (giorni) -> ");
  scanf("%d", &Ndays);
  NO=N-NL;
  NP=NO-Ndays;
  printf("\n Punti previsti: %d\n", NP);
  printf("\n Vicini usati  -> ");
  scanf("%d", &Nx);
  printf("\n Dare nome file output -> ");
  scanf("%s", file_out);
  printf("\n");
  y  = vector(1,Ntot);
  yl = matrix(1,NL,1,M);
```

```
  yo = matrix(1,NO,1,M);
  yp = matrix(1,NP,1,M);
  dists = vector(1,Nx);
  y_idx = ivector(1,Nx);  //indici degli Nx primi vicini
  for(int i=1; i<=Ntot; i++) fscanf(f_in, "%f", &y[i]);
  fclose(f_in);
  for(int i=1; i<=NL; i++) for(int j=1; j<=M; j++) yl[i][j]=y[i+(j-
1)*T];
  for(int i=1; i<=NO; i++) for(int j=1; j<=M; j++)
yo[i][j]=y[NL+i+(j-1)*T];
  for(int i=1; i<=NP; i++) for(int j=1; j<=M; j++) yp[i][j]=0.0;
  for(int i=1; i<=NP; i++) {
    int nn = PrimoVicino(yo, yl, i, NL, M);  //calcola il primo
vicino di yo[i]
    PrimiVicini(yl, y_idx, nn, NL, M, Nx);  //calcola i primi vicini
di yl[nn]
    for(int ii=1; ii<=Nx; ii++) dists[ii]=Dist(yo,yl,i,y_idx[ii],M);
    Weight(dists, Nx);
    for(int j=1; j<=M; j++) {
      for(int k=1; k<=Nx; k++) {
        yp[i][j]+=dists[k]*yl[y_idx[k]+Ndays][j];
      }
    }
  }
  for(int i=1; i<=NP; i++)
    for(int j=1; j<=M; j++)
      eqm+=fabs(yo[i][j]-yp[i][j]);
  eqm=eqm/NP;
  f_out = fopen(file_out, "w");
   fprintf(f_out, "%d\t%f\n", Ndays, eqm);
  for(int i=1; i<=NP; i++)
    fprintf(f_out, "%f\t%f\n", yo[i][M], yp[i][M]);
  fclose(f_out);
  free_vector(y,1);
  free_vector(dists,1);
  free_matrix(yl,1,1);
  free_matrix(yo,1,1);
  free_matrix(yp,1,1);
  free_ivector(y_idx,1);
}
```

Riferimenti bibliografici

H. D. I. Abarbanel, R. Brown, J. J. Sidorowich and L. S. Tsimring, (1993). *The Analysis of Observed Chaotic Data in Physical Systems.* Rev. Mod. Phys., **65**, 4, 1331.

H. D. I. Abarbanel (1995). *Analysis of Observed Chaotic Data.* Springer-Verlag, New York.

J. R. Buchler, (1997). *Search for Low-Dimensional Chaos in Observational Data.* International School of Physics "Enrico Fermi", Course CXXXIII. Eds. G. C. Castagnoli and A. Provenzale.

M. Ceschia, (1998). *Comunicazione privata.*

K. M. Cuomo and A. V. Oppenheim, (1993). *On acoustic, speech and signal processing.* Proc. Int. Conf., **3**, 137.

I. Daubechies, (1988). *Orthonormal bases of compactly supported wavelets.* Commun. Pure Appl. Math., **16**, 901.

D. Donoho, (1993). *Nonlinear wavelet methods for recovery of signals, densities, and spectra from indirect and noisy data.* Different Perspectives on Wavelets, Proceeding of Symposia in Applied Mathematics. I. Daubechies ed. Amer. Math. Soc., Providence, R.I., **47**, 173.

A. M. Fraser and H. L. Swinney, (1986). *Independent coordinates for strange attractors from mutual information.* Phys. Rev. A, **33**, 1134.

P. Grassberger, (1983), *On the fractal dimension of the Hénon attractor.* Physics Letters **97A**, 224.

P. Grassberger and I. Procaccia, (1983). *Characterization of strange attractors.* Phys. Rev. Letters, **50**, 346.

P. Grassberger, R. Hegger, H. Kantz, C. Schaffrath and T. Schreiber, (1993). *On noise reduction methods for chaotic data*. Chaos, **3**, 127.

P. Grassberger, (1986). *Do climatic attractors exist?* Nature, **323**, 609.

G. Kaiser, (1994). *A Friendly Guide to Wavelets.* Birkhäuser, Boston.

E. J. Kostelich and T. Schreiber, (1993). *Noise reduction in chaotic time-series data: A survey of common methods.* Phys. Rev. E, **48**, 1752.

A. Malignani, (1939). *Della temperatura atmosferica e dei modi di determinarla.* Tip. G. B. Doretti, Udine.

S. Mallat, (1989). *A theory for multiresolution signal decomposition: The wavelet representation.* IEEE Tran. Pattern Anal. Mach. Intel., **11**, 674.

R. Mañé, D. Rand and L. S. Young (1981), editors. *Dynamical Systems and Turbolence.* Page 230, Springer, Berlin.

C. Nicolis and G. Nicolis (1984), *Is there a climatic attractor?* Nature, **311**, 529.

C. Nicolis and G. Nicolis (1987), *Evidence for climatic attractors.* Nature, **326**, 523.

A. R. Osborne, A. Provenzale, (1992). *Finite correlation dimension for stochastic systems with power-law spectra.* Physica D, **35**, 357.

W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery (1992). *Numerical Recipes in C The Art of Scientific Computing.* Cambridge University Press, Cambridge, MA.

A. Provenzale, L. A. Smith, R. Vio and G. Murante (1992). *Distinguishing between low-dimensional dynamics and randomness in measured time series.* Physica D, **58**, 31.

H. Peitgen, H. Jürgens, and D. Saupe (1992). *Chaos and Fractals New Frontiers of Science*. Springer-Verlag, New York.

A. Rényi,(1970). *Probability Theory*. North-Holland, Amsterdam.

M. T. Rosenstein, J. J. Collins and C. J. De Luca, (1993). *A practical method for calculating largest Lyapunov exponents from small data sets.* Physica D, **65**, 117.

M. T. Rosenstein, J. J. Collins and C. J. De Luca, (1994). *Reconstruction expansion as a geometry-based framework for choosing proper delay times.* Physica D, **73**, 82.

N. Saito, (1994). *Simultaneous noise suppression and signal compression using a library of orthonormal bases and the minimum description length criterion.* In: *Wavelets in Geophysics*, Foufoula-Georgiou and Kumar (eds.), Academic Press.

T. Sauer, (1992). *A noise reduction method for signals from nonlinear system.* Physica D, **58**, 193.

S. Sato, M. Sano and Y. Sawada, (1987). *Practical methods of measuring the generalized dimension of the largest Lyapunov exponent in high dimensional chaotic systems.* Phys. Rev. Lett., **55**, 1082.

F. Takens, (1981). *Detecting strange attractors in turbolence.* In D. Rand and L. S. Young editors. *Dynamical Systems and Turbolence.* Page 230, Springer, Berlin.

J. Theiler (1986). *Spurious dimension from correlation algorithms applied to limited time-series data.* Phys. Rev. A, **34**, 2427.

J. M. T. Thompson and H. B. Stewart (1986). *Nonlinear Dynamics and Chaos*. John Wiley and Sons, Chichester.

A. A. Tsonis and J. B. Elsner (1988), *The weather attractor over very short timescales.* Nature, **333**, 545.

S. Venkadesan, M. C. Valsakumar, K. P. N. Murty and S. Rajasekar, (1996). *Evidence for chaos in an experimental time series from serrated plastic flow*. Phys. Rev. E, **54**, 611.