Chapter 1

NONLINEAR DYNAMICS TIME SERIES ANALYSIS

Bruce Henry, Nigel Lovell, and Fernando Camacho

1. INTRODUCTION

Much of what is known about physiological systems has been learned using linear system theory. However, many biomedical signals are apparently random or aperiodic in time. Traditionally, the randomness in biological signals has been ascribed to noise or interactions between very large numbers of constituent components.

One of the most important mathematical discoveries of the past few decades is that random behavior can arise in deterministic nonlinear systems with just a few degrees of freedom. This discovery gives new hope to providing simple mathematical models for analyzing, and ultimately controlling, physiological systems.

The purpose of this chapter is to provide a brief pedagogic survey of the main techniques used in nonlinear time series analysis and to provide a MATLAB tool box for their implementation. Mathematical reviews of techniques in nonlinear modeling and forecasting can be found in Refs. 1–5. Biomedical signals that have been analyzed using these techniques include heart rate [6–8], nerve activity [9], renal flow [10], arterial pressure [11], electroencephalogram [12], and respiratory waveforms [13].

Section 2 provides a brief overview of dynamical systems theory including phase space portraits, Poincaré surfaces of section, attractors, chaos, Lyapunov exponents, and fractal dimensions. The forced Duffing–Van der Pol oscillator (a ubiquitous model in engineering problems) is investigated as an illustrative example. Section 3 outlines the theoretical tools for time series analysis using dynamical systems theory. Reliability checks based on forecasting and surrogate data are also described. The time series methods are illustrated using data from the time evolution of one of the dynamical variables of the forced Duffing–Van der Pol oscillator. Section 4 concludes with a discussion of possible future directions for applications of nonlinear time series analysis in biomedical processes.

2. DYNAMICAL SYSTEMS THEORY

2.1. Deterministic Chaos

A *dynamical system* is any system that evolves in time. Dynamical systems whose behavior changes continuously in time are mathematically described by a coupled set of first-order autonomous ordinary differential equations

$$\frac{d\vec{x}(t)}{dt} = \vec{F}(\vec{x}(t), \vec{\mu}) \tag{1}$$

The components of the vector $\vec{x}(t)$ are the dynamical variables of the system, the components of the vector $\vec{\mu}$ are parameters, and the components of the vector field $\vec{F}$ are the dynamical rules governing the behavior of the dynamical variables. There is no loss of generality in the restriction to *autonomous systems*, where $\vec{F}$ is not an explicit function of t, since a nonautonomous system in $\mathbb{R}^n$ can be transformed into an autonomous system in $\mathbb{R}^{n+1}$. If the vector field is *affine*, i.e.,

$$\vec{F}(\vec{x}, \vec{\mu}) = \underline{\underline{A}}(\vec{\mu})\vec{x} + \vec{b}(\vec{\mu}) \tag{2}$$

for some constant matrix $\underline{\underline{A}}$ and vector $\vec{b}$, then the dynamical system is said to be *linear*. Otherwise it is *nonlinear*. In linear dynamical systems any linear combination of solutions is also a solution.

An example of a nonlinear dynamical system with numerous applications in engineering is the single well-forced Duffing–Van der Pol oscillator [14]

$$\frac{d^2y}{dt^2} - \mu(1 - y^2)\frac{dy}{dt} + y^3 = f\cos\omega t \tag{3}$$

where μ, f, ω are parameters. This second-order nonautonomous equation can be written as the first-order system

$$\frac{dx_1}{dt} = x_2 \tag{4}$$

$$\frac{dx_2}{dt} = \mu(1 - x_1^2)x_2 - x_1^3 + f\cos x_3 \tag{5}$$

$$\frac{dx_3}{dt} = \omega \tag{6}$$

by defining dynamical variables $x_1 = y$, $x_2 = dy/dt$, $x_3 = \omega t$.

Dynamical systems whose behavior changes at discrete time intervals are described by a coupled set of first-order autonomous difference equations

$$\vec{x}(n+1) = \vec{G}(\vec{x}(n), \vec{\mu}) \tag{7}$$

In this equation $\vec{G}$ describes the dynamical rules and time is represented by the integer n. A discrete dynamical system may be obtained from a continuous dynamical system (1) by sampling the solution of the continuous dynamical system at regular time intervals T—the dynamical rule relating successive sampled values of the dynamical variables is called a *time T map*, and (2) by sampling the solution of the continuous dynamical system in $\mathbb{R}^n$ at successive transverse intersections with a surface of section of dimension $\mathbb{R}^{n-1}$—the dynamical rule relating successive sampled values of the dynamical variables is called a *Poincaré map* or a *first return map*. For example, in the forced Duffing–Van der Pol oscillator a surface of section could be defined by $x_3 = \theta_0$ where

$\theta_0 \in (0, 2\pi)$ is a constant. In this case the Poincaré map is equivalent to a time T map with $T = 2\pi/\omega$.

Under modest smoothness assumptions about the dynamical rules, the solutions of dynamical systems are unique and the dynamical system is *deterministic*; that is, the state of the dynamical system for all times is uniquely determined by the state at any one time. The *existence* of unique solutions does not necessarily mean that *explicit algebraic representations* exist. However, if explicit algebraic solutions do exist and they can be used to predict the future behavior of the system for all initial conditions, then the system is said to be *integrable*. All *linear* dynamical systems are integrable. Explicit solutions can be constructed for linear systems by first transforming to a new set of dynamical variables in which the governing equations become decoupled.

One of the surprising and far-reaching mathematical discoveries of the past few decades has been that the solutions of deterministic *nonlinear* dynamical systems may be as random (in a statistical sense) as the sequence of heads and tails in the toss of a fair coin [15]. This behavior is called *deterministic chaos*. The discovery of deterministic chaos is surprising because randomness has been traditionally associated with unknown external disturbances (noise). What makes the discovery far reaching is that most dynamical systems are nonlinear and most nonlinear systems have random solutions. Deterministic chaos has immediate ramifications for constructing mathematical models for systems characterized by random signals. A fundamental question in this regard is: Are all random signals equally random? It turns out that they are not. Random signals generated by noise are fundamentally different from random signals generated by deterministic dynamics with small numbers of dynamical variables. The difference is not revealed by statistical analysis but is instead revealed by dynamical analysis based on phase space reconstruction.

2.2. Phase Space—Attractors

Phase space is an abstract mathematical space spanned by the dynamical variables of the system. The state of the dynamical system at a given instant in time can be represented by a point in this phase space. If there are n dynamical variables, then the state at a given time can be represented by a point in the Euclidean space $\mathbb{R}^n$. As the dynamical variables change their values in time, the representative point traces out a path in the phase space—a continuous curve in the case of a continuous dynamical system and a sequence of points in the case of a discrete dynamical system.

For an idealized simple pendulum there are two physical dynamical variables, the angular position θ and velocity $\dot{\theta}$, so the phase space can be taken to be $\mathbb{R}^2$. If the energy is conserved in this system and the angular oscillations are small, then $E = \frac{1}{2}\dot{\theta}^2 + \frac{1}{2}\theta^2$ constrains the phase space variables to lie on a circle. The radius of the circle is determined by the system energy. A realistic pendulum dissipates energy to the surroundings via friction and air resistance. The phase space path in this case is a spiral in toward a final resting point called a fixed point attractor. The starting radius for the spiral again depends on the initial energy in the system, but the location of the fixed point attractor is independent of this starting energy. Most physical systems are dissipative and their long-time dynamical behavior can be described by an attractor in phase space. In Figure 1 phase space portraits are shown for the Duffing–Van der Pol oscillator, Eqs. 4–6 for initial conditions $x_1 = 1$, $x_2 = 0$, $x_3 = 0$, and four sets of parameters: (a) $\mu = 0.0, f = 0.0$; (b) $\mu = 0.2, f = 0.0$; (c) $\mu = 0.2, f = 1.0, \omega = 0.9$; and (d)

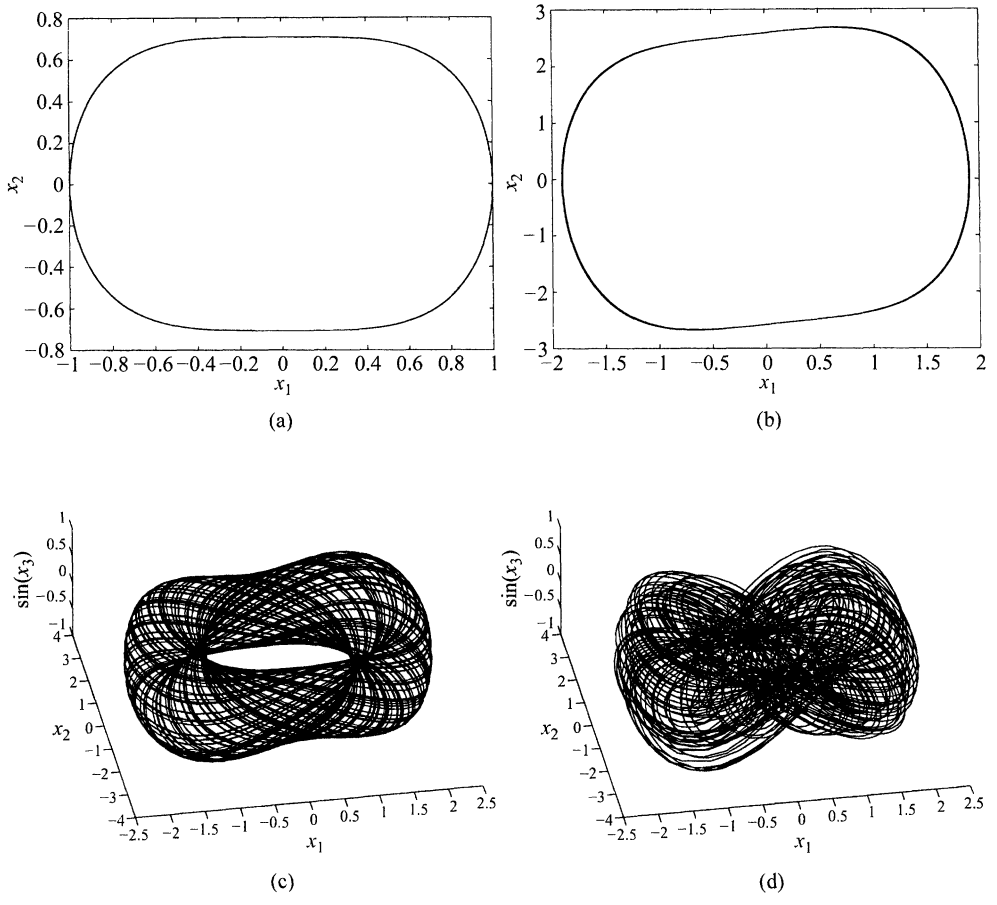


Figure 1 Phase space portraits for the Duffing–Van der Pol oscillator for parameters (a) $\mu = 0.0, f = 0.0$; (b) $\mu = 0.2, f = 0.0$; (c) $\mu = 0.2, f = 1.0, \omega = 0.90$; (d) $\mu = 0.2, f = 1.0, \omega = 0.94$.

$\mu = 0.2, f = 1.0, \omega = 0.94$. In cases (c) and (d) it is convenient to choose as coordinate axes the dynamical variables x_1, x_2, $\sin(x_3)$. The initial transient behavior is not shown in these figures.

For the parameters in case (a), there is no dissipation and the path is topologically similar to that of the idealized pendulum. The distortion away from a circle is due to nonlinear restoring forces. In case (b), dissipation is included ($\mu \neq 0$) and the path is a limit cycle attractor. In case (c), the dissipation ($\mu \neq 0$) is balanced by external forcing ($f \neq 0$) and another periodic orbit results [14]. The conditions in case (d) are very similar to the conditions in case (c) but the slightly different forcing frequency in case (d) results in a very different orbit—a chaotic strange attractor [14]. The Poincaré surface of section equivalent to the time $2\pi/\omega$ map for the two sets of parameters, case (c) and case (d), is shown in Figure 2a and b, respectively. Again, the initial transient behavior is not shown.

Appendix I contains MATLAB programs for numerically integrating three coupled differential equations (Appendix I.A), generating three-dimensional phase

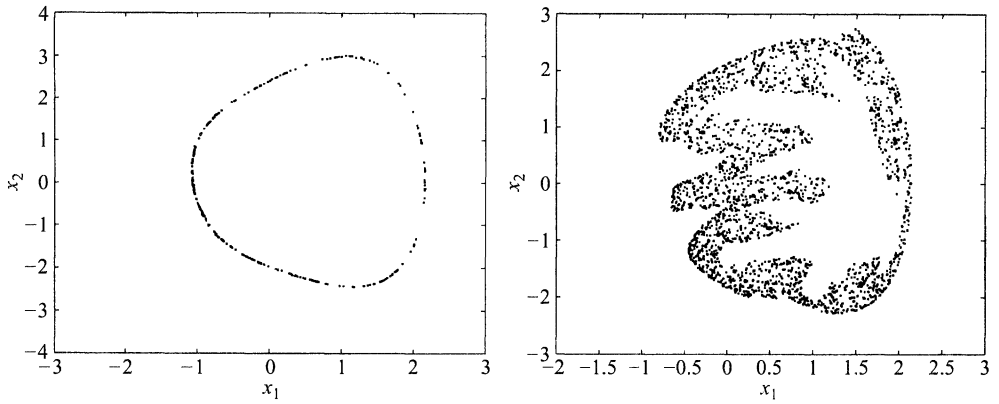


Figure 2 Poincaré surfaces of section for the Duffing–Van der Pol oscillator for parameters (a) $\mu = 0.2, f = 1.0, \omega = 0.9$; (b) $\mu = 0.2, f = 1.0, \omega = 0.94$.

space portraits (Appendix I.B), and constructing two-dimensional Poincaré surfaces of section (Appendix I.C).

The concepts of dissipation and attractors are defined more precisely for a *continuous* dynamical system as follows. A dynamical system is *conservative* if

$$\vec{\nabla} . \vec{F} = 0 \tag{8}$$

A dynamical system is *dissipative* if on average (where the average is over all initial conditions)

$$\vec{\nabla} . \vec{F} < 0 \tag{9}$$

In a dissipative dynamical system, phase space volume elements contract as the system evolves. In the forced Duffing–Van der Pol oscillator,

$$\vec{\nabla} . \vec{F} = -\mu(x_1^2 - 1) \tag{10}$$

so that the system is dissipative for $x_1^2 > 1$; $\mu > 0$.

A point $\vec{x}$ is a *fixed point* (or equilibrium point) if

$$\vec{F}(\vec{x}) = 0 \tag{11}$$

A phase space trajectory $\vec{x}(t)$ is *periodic* if

$$\vec{x}(t) = \vec{x}(t + T) \tag{12}$$

for some nonzero T. The period is the minimum nonzero T for which Eq. 12 holds. It is convenient to represent a phase space path by

$$\vec{x}(t) = \phi^{t,t_0}(\vec{x}_0) \tag{13}$$

where ϕ is the flow that maps an initial point $\vec{x}_0 \in \mathbb{R}^n$ and time $t \in \mathbb{R}$ onto the solution $\vec{x}(t)$. An ω-limit point of $\vec{x}_0$ is a phase space point $\vec{x}$ such that

$$\lim_{t\to\infty} \phi^{t,t_0}(\vec{x}_0) \to \vec{x} \tag{14}$$

An ω-limit set is a set of ω-limit points corresponding to a set of initial points $\vec{X}_0 = \{\vec{x}_0^{(1)}, \vec{x}_0^{(2)}, \ldots, \vec{x}_0^{(k)}\}$. An *attracting set* is an ω-limit set to which all orbits starting in the neighborhood of the set of initial points $\vec{X}_0$ tend as $t \to \infty$. An *attractor* is an attracting set that contains a dense orbit. A system must be dissipative in order to have an attractor. Since phase space volume elements contract in time in dissipative systems, it follows that attractors must occupy zero volume in phase space. A *limit cycle* attractor is a periodic attractor.

A *strange attractor* is an aperiodic attractor with the additional properties that (1) phase space paths through all points on the attractor diverge on average at an exponential rate and (2) the dimension of the set of points comprised by the attractor is not an integer.

2.3. Lyapunov Exponents

Lyapunov exponents quantify the average exponential separation between nearby phase space trajectories. An exponential divergence of initially nearby trajectories in phase space coupled with folding of trajectories (to ensure that solutions remain finite) is the generic mechanism for generating deterministic randomness and unpredictability. Indeed, the existence of a positive Lyapunov exponent for almost all initial conditions in a bounded dynamical system is a widely used definition of deterministic chaos.

Let $\vec{x}_0(t)$ denote a reference trajectory passing through $\vec{x}_0(0)$ at time $t = 0$ and let $\vec{x}_1(t)$ denote a trajectory passing through $\vec{x}_1(0)$ at time $t = 0$. The (maximum) Lyapunov exponent $\lambda(\vec{x}_0)$ is defined with respect to the reference orbit $\vec{x}_0$ by [16,17]

$$\lambda(\vec{x}_0) = \lim_{t\to\infty} \lim_{\|\Delta\vec{x}(0)\|\to 0} \frac{1}{t} \log \frac{\|\Delta\vec{x}(t)\|}{\|\Delta\vec{x}(0)\|} \tag{15}$$

where $\|\Delta\vec{x}(0)\|$ is the Euclidean distance between the trajectories $\vec{x}_0(t)$ and $\vec{x}_1(t)$ at an initial time $t = 0$ and $\|\Delta\vec{x}(t)\|$ is the Euclidean distance between the trajectories $\vec{x}_0(t)$ and $\vec{x}_1(t)$ at a later time t. In this definition $\vec{x}_1(t)$ can be any trajectory that is initially infinitesimally close to $\vec{x}_0(0)$ at time $t = 0$. The correspondence between sensitivity to initial conditions and a positive Lyapunov exponent is obvious in the rearrangement

$$\|\Delta\vec{x}(t)\| \sim \|\Delta\vec{x}(0)\| e^{\lambda t} \tag{16}$$

A dynamical system in $\mathbb{R}^m$ has associated with it m Lyapunov exponents

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m \tag{17}$$

To define the full set of exponents, consider an infinitesimal m-dimensional sphere of initial conditions that is anchored to a reference trajectory. As the sphere evolves, it becomes deformed into an ellipsoid. Let $p_i(t)$ denote the length of the ith principal axis, ordered from most rapidly growing to least rapidly growing. Then [16,18]

$$\lambda_i = \lim_{t\to\infty} \frac{1}{t} \log\left(\frac{p_i(t)}{p_i(0)}\right) \qquad i = 1, 2, \ldots, m \tag{18}$$

defines the set of Lyapunov exponents ordered from largest to smallest. Volume elements in phase space evolve in time as [19]

$$V(t) \sim V(0) \exp\left(\sum_i^m \lambda_i t\right) \tag{19}$$

The sum of the Lyapunov exponents is equal to the divergence of the vector field; i.e.,

$$\sum_i^m \lambda_i = \vec{\nabla} . \vec{F} \tag{20}$$

Thus, in dissipative systems the average of the sum of the Lyapunov exponents is negative. Furthermore, for bounded trajectories that do not approach a fixed point at least one of the Lyapunov exponents is zero.

In numerical computations of Lyapunov exponents the limit $\|\Delta \vec{x}(0)\| \to 0$ can be effectively realized by evolving $\Delta \vec{x}(t)$ with the linear variational equation [20]

$$\frac{d\Delta \vec{x}(t)}{dt} = D_x \vec{F}\left(\vec{x}_0(t)\right)\Delta \vec{x}(t) \tag{21}$$

In this equation $\vec{x}_0(t)$ is the reference orbit obtained by integrating the original dynamical system and $D_x \vec{F}$ is a matrix with components $\partial F_i/\partial x_j$ evaluated along the reference orbit. If the system has a positive Lyapunov exponent, then direct numerical integration of the linear variational equation will eventually lead to numerical overflow. This problem can be avoided by renormalizing the solution of the linear variational equation at a periodic intervals τ. The maximum Lyapunov exponent is then equivalently given by [20]

$$\lambda_{\max} = \lim_{n\to\infty} \frac{1}{(n+1)\tau} \sum_{j=0}^{n} \log \|\Delta \vec{x}_j(\tau)\| \tag{22}$$

where $\Delta \vec{x}_j(\tau)$ are solutions of the variational equations for renormalized initial vectors

$$\Delta \vec{x}_j(0) = \frac{\Delta \vec{x}_{j-1}(\tau)}{|\Delta \vec{x}_{j-1}(\tau)|} \tag{23}$$

A similar renormalization using the Gram-Schmidt reorthonormalization scheme can be employed to measure the full set of Lyapunov exponents [21,22]. In Table 1 the full

TABLE 1 Lyapunov Exponents for the Duffing–Van der Pol Oscillator for Parameters

Parameters[a]	λ_1	λ_2	λ_3
(a)	0.000	0.000	
(b)	0.000	−0.133	
(c)	0.000	0.000	−0.049
(d)	0.0254	0.000	−0.0285

[a] (a), $\mu = 0.0, f = 0.0$; (b), $\mu = 0.2, f = 0.0$; (c), $\mu = 0.2, f = 1.0, \omega = 0.90$; (d), $\mu = 0.2, f = 1.0, \omega = 0.94$.

set of Lyapunov exponents is listed for each of the four sets of parameters used in Figure 1. Note that case (d) has a positive Lyapunov exponent and the sum of the exponents is negative. This is consistent with chaotic dynamics on a strange attractor.

A MATLAB code for measuring the Lyapunov exponents for three coupled differential equations is listed in Appendix I.D.

2.4. Fractal Dimensions

A geometrical object can be fully represented by a set of points in a Euclidean space $\mathbb{R}^m$ provided that m is sufficiently large to be able to uniquely locate the position of each point in the object. Each set in $\mathbb{R}^m$ has assigned to it a topological dimension d that is an integer in the range $[0, m]$. If the set is all of $\mathbb{R}^m$, then $d = m$. In Euclidean geometry, points have dimension $d = 0$, lines have dimension $d = 1$, plane surfaces have dimension $d = 2$, solids have dimension $d = 3$, etc.

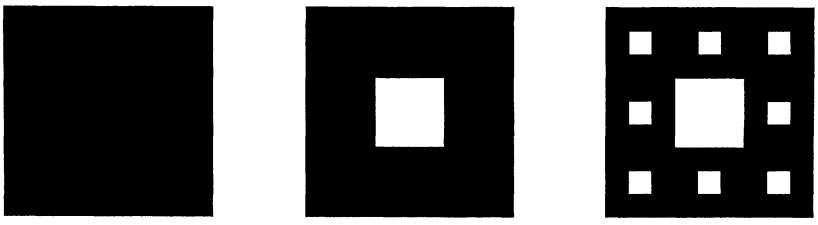
A *fractal dimension* D is any dimension measurement that allows noninteger values [23]. A *fractal* is a set with a noninteger fractal dimension [23]. Standard objects in Euclidean geometry are not fractals but have integer fractal dimensions $D = d$. The primary importance of fractals in dynamics is that strange attractors are fractals and their fractal dimension D is simply related to the minimum number of dynamical variables needed to model the dynamics of the strange attractor.

The simplest way (conceptually) to measure the dimension of a set is to measure the *Kolmogorov capacity* (or box-counting dimension). In this measurement a set is covered with small cells (e.g., squares for sets embedded in two dimensions, cubes for sets embedded in three dimensions) of size ϵ. Let $M(\epsilon)$ denote the number of such cells that contain part of the set. The dimension is then defined as

$$D = \lim_{\epsilon \to 0} \frac{\log(M(\epsilon))}{\log\left(\frac{1}{\epsilon}\right)} \tag{24}$$

For n isolated points, $M(\epsilon) = n$ and $D = 0$; for a straight line of length L, $M(\epsilon) = L/\epsilon$ and $D = 1$; for a plane region of area A, $M(\epsilon) = A/\epsilon^2$ and $D = 2$. In practical applications the limit ϵ is not attainable. Instead, the number $M(\epsilon)$ is measured for a range of small values of ϵ and the dimension D is estimated as the slope of the straight line portion of the plot of $\log(M(\epsilon))$ versus $\log(1/\epsilon)$.

A mathematical example of a set with a noninteger fractal dimension is the Cantor set, which is defined as the limiting set in a sequence of sets. Consider the set in $\mathbb{R}^2$ defined by the following sequence of sets. At stage $k = 0$ (Figure 3a) let S_0 denote a square with sides of length l. At stage $k = 1$ (Figure 3b) divide the set S_0 into nine squares of uniform size and remove the middle square. The remaining set is labeled S_1.



(a) (b) (c)

Figure 3 First three stages in the construction of a Cantor set in $\mathbb{R}^2$; (a) stage $k = 0$; (b) stage $k = 1$; (c) stage $k = 2$.

At stage $k = 2$ (Figure 3c) divide each remaining square in S_1 into nine squares of uniform size and remove the middle squares. This new set is labeled S_2. The process of subdividing and removing is continued iteratively to obtain a sequence of sets $S_0, S_1, S_2, \ldots$ and the Cantor set is defined as the limiting set

$$S = \lim_{n \to \infty} S_n \tag{25}$$

It is straightforward to measure the Kolmogorov capacity for this Cantor set. At stage $k = 0$ the set S_0 is covered with one square of size a. Thus for $k = 0$; $\epsilon = a$ and $M(\epsilon) = 1$. At stage $k = 1$ the set S_1 is covered with eight squares of size $a/3$. Thus for $k = 1$; $\epsilon = a/3$ and $M(\epsilon) = 8$. At stage $k = 2$ the set S_2 is covered with 64 squares of size $a/9$. Thus for $k = 2$; $\epsilon = a/9$ and $M(\epsilon) = 64$. For general k it follows by induction that the set S_k is covered with $\epsilon = a/3^k$ and $M(\epsilon) = 8^k$. The fractal dimension of the limiting Cantor set is thus

$$D = \lim_{k \to \infty} \frac{\log(8^k)}{\log\left(\frac{a}{3^k}\right)} \tag{26}$$

$$= 1.892 \tag{27}$$

The fractal dimension less than two means that this Cantor set does not fill an area in $\mathbb{R}^2$.

When computing the box-counting dimension, a box is counted whenever it contains part of the set. This counting does not differentiate between whether a box contains many points of the set or few points of the set. More elaborate dimension measurements are available that take into account inhomogeneities or correlations in the set. The *dimension spectrum* defined by Hentschel and Procaccia [24],

$$D_q = \lim_{r \to 0} \frac{1}{q-1} \frac{\log \sum_{i=1}^{M(r)} p_i^q}{\log r}, \qquad q = 0, 1, 2, \ldots \tag{28}$$

provides a set of fractal dimension measurements that take into account higher order correlations as q is increased. In the dimension spectrum, $M(r)$ is the number of m-dimensional cells of size r (e.g., hypercubes of side r) needed to cover the set and $p_i = N_i/N$ is the probability of finding a point of the set in hypercube i; N is the total number

of points in the set and N_i is the number of points of the set in hypercube i. It can be readily seen that the box-counting dimension is equivalent to D_0.

The dimension D_1 is called the *information dimension*. This is defined by taking the limit $q \to 1$, i.e.,

$$D_1 = \lim_{q \to 1} D_2 \tag{29}$$

$$= \lim_{r \to 0} \frac{\sum_{i=1}^{M(r)} p_i \log p_i}{\log r} \tag{30}$$

The information dimension has also been related to the Lyapunov exponents through a conjecture of Kaplan and Yorke [25]:

$$D_1 = j + \frac{\sum_{i=1}^{j} \lambda_i}{|\lambda_{j+1}|} \tag{31}$$

In Eq. 31, λ_i are the Lyapunov exponents of the attractor ordered from largest to smallest and $\sum_{i=1}^{j} \lambda_i \geq 0$; $\sum_{i=1}^{j+1} \lambda_i < 0$. As an example consider the strange attractor in the Duffing–Van der Pol oscillator with parameters $\mu = 0.2$, $f = 1.0$, $\omega = 0.94$, Figure 1d. The Lyapunov information dimension for this case is, from the values in Table 1 (d), $D_1 = 2.84$. The noninteger value confirms that the attractor is a strange attractor.

The dimension D_2 is called the *correlation dimension*. This can be written as

$$D_2 = \lim_{r \to 0} \frac{\log C(r)}{\log r} \tag{32}$$

where

$$C(r) = \sum_{i=1}^{M(r)} p_i^2 \tag{33}$$

is the correlation sum, which is essentially (exact in the limit $N \to \infty$) the probability that two points of the set are in the same cell.

For a given set, the dimensions are ordered $D_0 \geq D_1 \geq D_2 \geq \cdots$. If the set is a homogeneous attractor then

$$p_i = \frac{1}{M} \tag{34}$$

and all dimensions, Eq. 28, are equal; otherwise the set is called a *multifractal*. The major difficulty in calculating D_q is the practical difficulty of covering the set with cells of very small size. In general, this requires too much computer storage and time to obtain convergence to a limit $r \to 0$.

When $q = 2$ the dimension estimate can be made computationally tractable by using an algorithm proposed by Grassberger and Procaccia [26]. This algorithm has

become the most widely used method for estimating fractal dimensions of experimental data sets.

2.5. Grassberger–Procaccia Algorithm

The Grassberger–Procaccia algorithm [26] is based on the following approximation: The probability that two points of the set are in the same cell of size r is approximately equal to the probability that two points of the set are separated by a distance ρ less than or equal to r. Thus $C(r)$ is approximately given by

$$C(r) \approx \frac{\sum_{i=1}^{N} \sum_{j>i} \Theta\left(r - \rho(\vec{x}_i, \vec{x}_j)\right)}{\frac{1}{2}N(N-1)} \tag{35}$$

where the Heaviside function is defined as

$$\Theta(s) = \begin{cases} 1 & \text{if } s \geq 0 \\ 0 & \text{if } s < 0 \end{cases} \tag{36}$$

The approximation in Eq. 35 is exact in the limit $N \to \infty$; however, this limit cannot be realized in practical applications. The limit $r \to 0$ used in the definition of D_2 is also not possible in practice. Instead, Procaccia and Grassberger propose the (approximate) evaluation of $C(r)$ over a range of values of r and then deduce D_2 from the slope of the straight line of best fit in the linear scaling region of a plot of $\log C(r)$ versus $\log r$.

The most common metric employed to measure the distance ρ in Eq. 35 is the Euclidean metric,

$$\rho(\vec{x}_i, \vec{x}_j) = \sqrt{\sum_{k=1}^{m} \left(x_i(k) - x_j(k)\right)^2} \tag{37}$$

However, other metrics have also been considered. In any case, the choice of metric should not affect the scaling of the correlation sum with r.

The reliability of estimating the slope in the linear scaling region is the most serious possible source of error in the Grassberger–Procaccia algorithm. Clearly, the linear scaling regime will be bounded above by the maximum separation distance between points in the set and bounded below by the minimum separation distance between points in the set. Essentially, the idea is to measure the slope for the smallest r values possible while avoiding the problems of sparse numbers when r is close to the minimum separation. One ad hoc scheme that has received some popularity is to plot $\log C(r)$ versus $\log r$ for a number of equally spaced values of $\log r$ between $\log r_{\min}$ and $\log r_{\max}$. Then deduce the slope of the straight line of best fit over the middle third of the vertical range of the plot. This method should be used with caution as it is possible that the middle third straddles two different straight line behaviors—noise and deterministic chaos. In particular, if the system contains noise on a scale r^* then for an m-dimensional embedding the correlation sum will scale as

$$C(r) \sim \begin{cases} r^m & \text{for } r < r^* \\ r^D & \text{for } r > r^* \end{cases} \tag{38}$$

Thus a plot of $\log C(r)$ versus $\log r$ will reveal a change of slope from m to D with the crossover at r^* providing an estimate of the level of noise in the system.

Figure 4 shows a plot of $\log C(r)$ versus $\log r$ for each of the phase space paths of the Duffing–Van der Pol oscillator shown in Figure 1. The straight lines of best fit using data across the domain from 10^{-2} to 10^{-1} are also shown. Estimates of the correlation dimension based on the slopes of these straight line portions are (a) $D_2 = 1.01$, (b) $D_2 = 1.01$, (c) $D_2 = 2.17$, (d) $D_2 = 2.67$.

A MATLAB code for implementing the Grassberger–Procaccia algorithm to measure the correlation dimension of a phase space trajectory is listed in Appendix I.E.

There have been several estimates of the minimum number of data points $N_{\min}$ required for estimates of D to be reliable using the Grassberger–Procaccia algorithm. A "rule of thumb" estimate due to Ruelle [27] is that

$$N_{\min} = 10^{(D/2)} \tag{39}$$

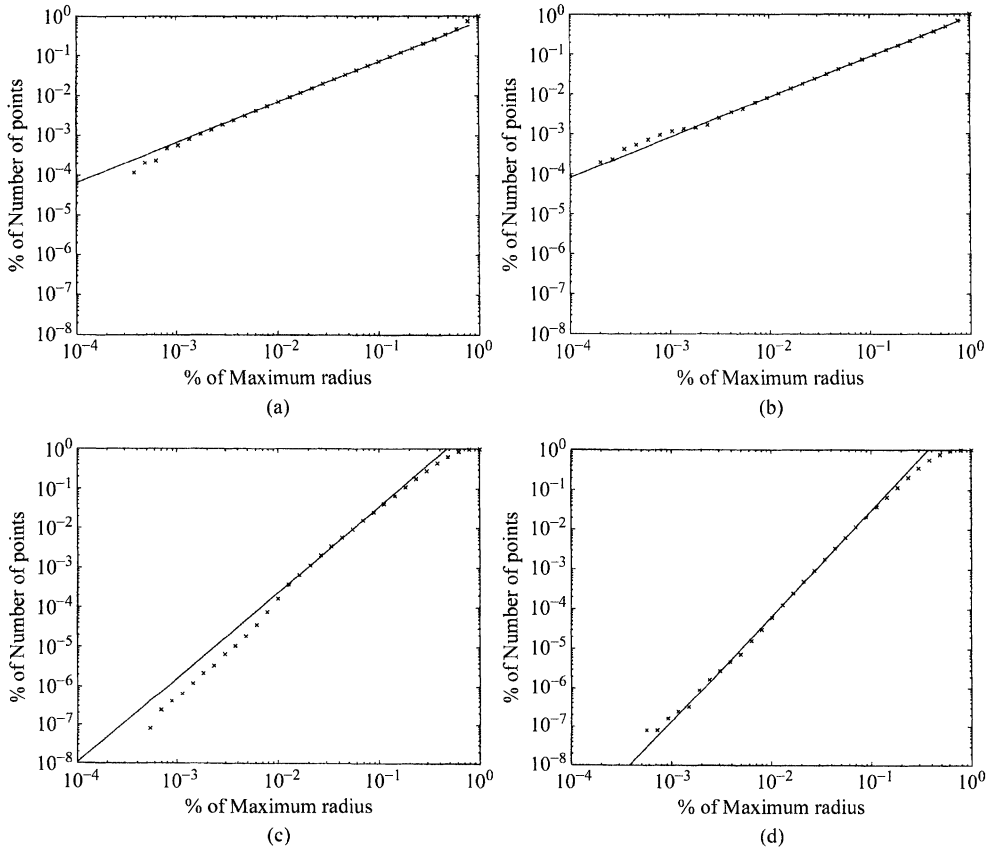


Figure 4 Plots of the logarithm of the correlation sum versus the logarithm of the separation distance for phase space trajectories of the Duffing–Van der Pol oscillator for parameters (a) $\mu = 0.0$, $f = 0.0$; (b) $\mu = 0.2$, $f = 0.0$; (c) $\mu = 0.2, f = 1.0$, $\omega = 0.90$; (d) $\mu = 0.2, f = 1.0$, $\omega = 0.94$.

Thus, estimates of $D \geq 2 \log_{10} N$ obtained from time series analysis should be regarded as unreliable. A simple derivation of this estimate is as follows: The correlation sum scales as

$$C(r) \sim Ar^D \tag{40}$$

Assume that this scaling applies up to the maximum separation distance $r_{\max}$. Then

$$A \sim \frac{C(r_{\max})}{r_{\max}^D} \tag{41}$$

but at the limiting separation

$$C(r_{\max}) \sim \tfrac{1}{2} N^2 \tag{42}$$

Combining the preceding three equations now yields

$$C(r) \sim \frac{N^2}{2} \left(\frac{r}{r_{\max}} \right)^D \tag{43}$$

Clearly, the correlation sum is bounded below by $C(r_{\min}) = 1$, hence $C(r) > 1$ and

$$2 \log N > D(\log r_{\max} - \log r) \tag{44}$$

The *Ruelle conjecture*, Eq. 39, now follows immediately from the reasonable expectation that the linear scaling regime in the log-log plot (if it exists) should persist over at least one decade in the range of r. The theoretical basis of the Ruelle conjecture has been questioned and other theoretical requirements on the sample size have been proposed [28,29]. However the rule of thumb, Eq. 39, has been found to be relevant in many experimental studies [27]. Moreover, all theoretical results for requirements on sample size reveal an exponential growth with dimension.

3. TIME SERIES ANALYSIS

The techniques in this section are illustrated using data from numerical integrations of the Duffing–Van der Pol oscillator Eqs. 4–6 with initial conditions $x_1 = 1$, $x_2 = 0$, $x_3 = 0$ and the four sets of parameters (a) $\mu = 0.0, f = 0.0$; (b) $\mu = 0.2, f = 0.0$; (c) $\mu = 0.2$, $f = 1.0$, $\omega = 0.90$; (d) $\mu = 0.2, f = 1.0$, $\omega = 0.94$. A time series y_n is constructed from the numerical solution for the single dynamical variable $x_1(t)$ sampled at intervals $\Delta t = 0.1$, i.e., $y_n = x_1(n\Delta t)$. The results in this section based on the time series analysis of a single variable can be compared with the results in Section 2 based on the direct analysis of the evolution of all dynamical variables. Time series for the Duffing–Van der Pol oscillator with this set of parameters are shown in Figure 5.

3.1. Power Spectrum and Autocorrelation

The power spectrum reveals periodic components of a signal and it should always be employed in time series analysis whether the primary analysis is statistical or dyna-

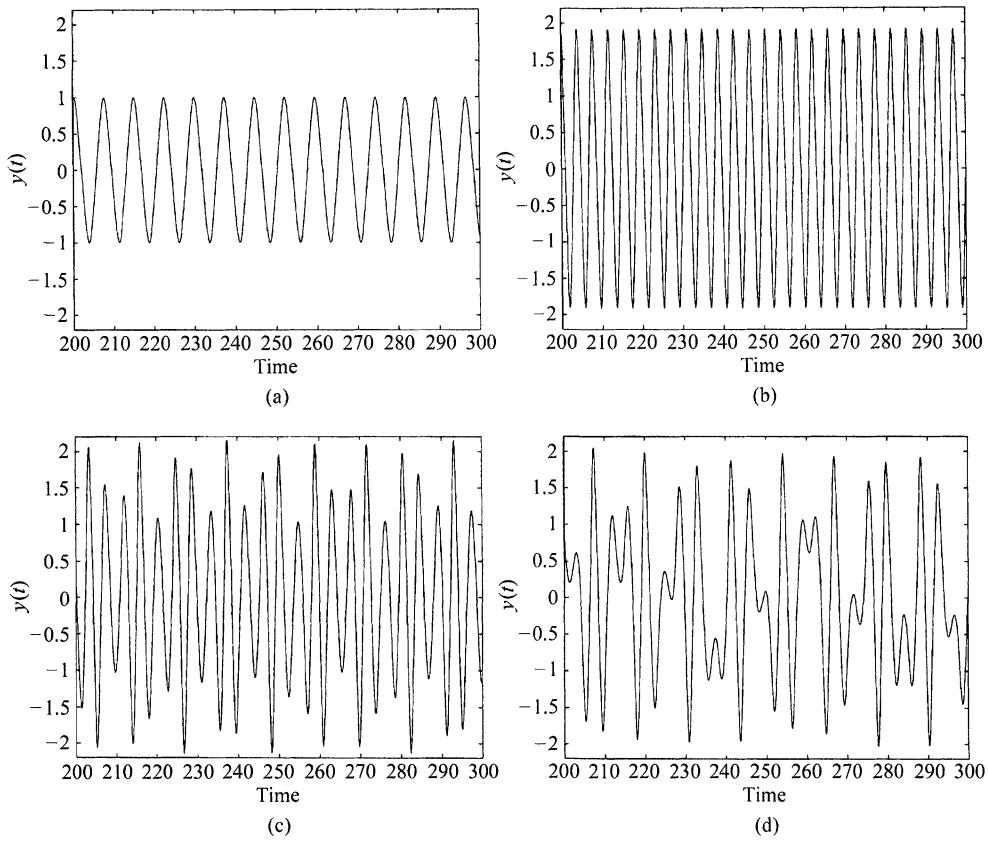


Figure 5 Time series data for the Duffing–Van der Pol oscillator for parameters (a) $\mu = 0.0, f = 0.0$; (b) $\mu = 0.2, f = 0.0$; (c) $\mu = 0.2, f = 1.0, \omega = 0.90$; (d) $\mu = 0.2, f = 1.0, \omega = 0.94$.

mical. If the signal is periodic then the power spectrum will consist of discrete lines, whereas in a stochastic signal the power will be spread over a continuous range of frequencies. Consider a time series

$$y_n = y(n\Delta t), \qquad n = 1, 2, \ldots, N \tag{45}$$

The discrete Fourier transform is defined by

$$Z_m = \frac{1}{N} \sum_{n=0}^{N-1} y_n \exp(-i2\pi(m-1)(n-1)/N) \tag{46}$$

and the power spectrum is defined as

$$P_m = |Z_m|^2 = X_m^2 + Y_m^2 \tag{47}$$

where X and Y are the real and imaginary parts of Z. Each value of m for which there is a peak in the power spectrum corresponds to a frequency component

$$f_m = \frac{m}{N\Delta t} \tag{48}$$

in the original time series.

The autocorrelation function also provides a diagnostic tool for discriminating between periodic and stochastic behavior. In a periodic signal the autocorrelation is periodic, whereas in a stochastic signal the autocorrelation will be irregular. The autocorrelation function is defined by

$$c_j = \frac{1}{N}\sum_{i=1}^{N} y_i y_{i+j} \tag{49}$$

where periodic boundary conditions

$$y_{N+k} = y_k \tag{50}$$

are imposed to extend the times series beyond y_N. This function provides a simple quantitative measure of the linear correlation between data points.

Fourier transforms of the time series data $y_n = x_1(n\Delta t)$ in the Duffing–Van der Pol oscillator are shown for four sets of parameters in Figure 6. The autocorrelation functions for the same sets of data are shown in Figure 7. The "grassy appearance" of the Fourier transform in Figure 6d and the aperiodicity of the autocorrelation function in Figure 7d are characteristic of a chaotic signal, whereas the sharp peaks in Figure 6a–c and the periodicities in Figure 7a–c are characteristic of periodic signals. The horizontal lines in Figure 7 show the value $1/e$.

3.2. Phase Space Reconstruction

The essential problem in nonlinear time series analysis is to determine whether or not a given time series is a deterministic signal from a low-dimensional dynamical system. If it is, then further questions of interest are: What is the dimension of the phase space supporting the data set? Is the data set chaotic?

The key to answering these questions is embodied in the method of phase space reconstruction [30], which has been rigorously justified by the embedding theorems of Takens [31] and Sauer et al. [32,33]. Takens' embedding theorem asserts that if a time series is one component of an attractor that can be represented by a smooth d-dimensional manifold (with d an integer) then the topological properties of the attractor (such as dimension and Lyapunov exponents) are equivalent to the topological properties of the embedding formed by the m-dimensional phase space vectors

$$\vec{X}_i = (y(i\Delta t),\ y(i\Delta t + \tau),\ y(i\Delta t + 2\tau), \ldots,\ y(i\Delta t + (m-1)\tau)) \tag{51}$$

whenever $m \geq 2d + 1$. In Eq. 51 τ is called the delay time and m is the embedding dimension. Different choices of τ and m yield different reconstructed trajectories. Takens' theorem has been generalized by Sauer et al. to the case where the attractor

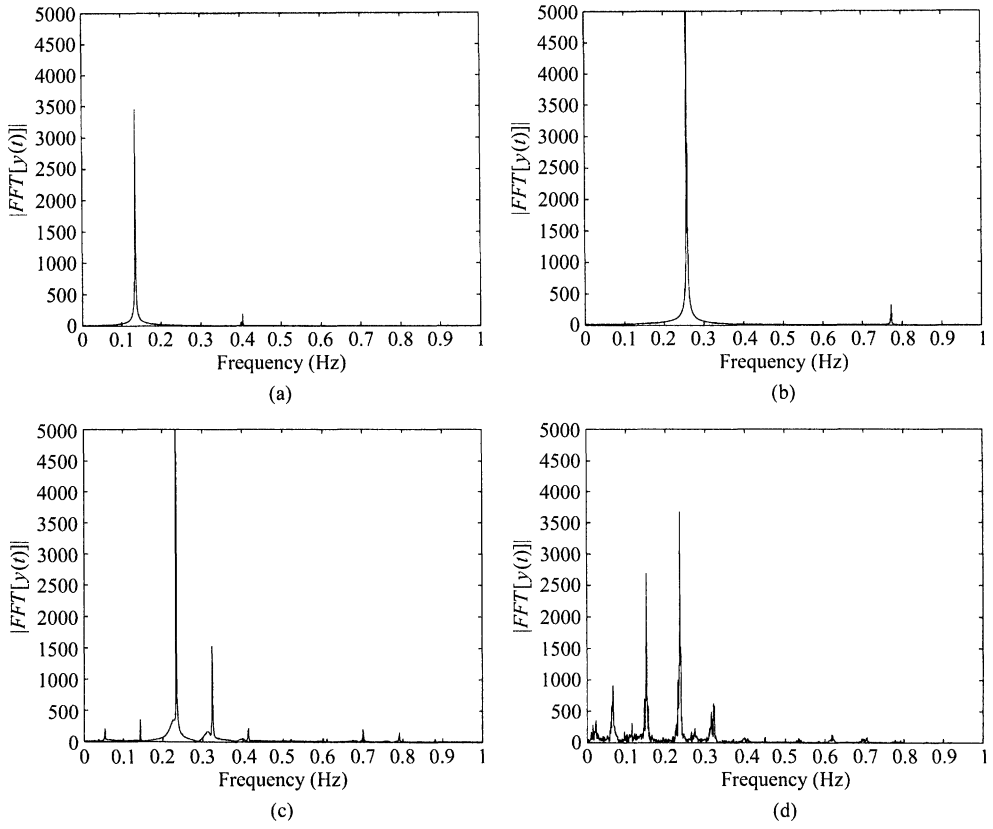


Figure 6 Absolute value Fourier transforms for time series data from the Duffing–Van der Pol oscillator for parameters (a) $\mu = 0.0, f = 0.0$; (b) $\mu = 0.2, f = 0.0$; (c) $\mu = 0.2, f = 1.0; \omega = 0.90$; (d) $\mu = 0.2, f = 1.0, \omega = 0.94$.

is a strange attractor with a fractal dimension D. The embedding of a strange attractor using time delay coordinates is one to one if $m \geq 2D + 1$.

There are several technical issues to bear in mind in nonlinear time series analysis. Foremost is the quality of the time series itself. If there is a simple deterministic rule governing the evolution of the time series, then the time interval between data points should be sufficiently small, the length of the data should be sufficiently long, and the level of noise should be sufficiently low to allow detection of the deterministic dynamics and subsequent forecasting. In biomedical signals the sampling rate, the signal length, and the noise level are typically limited by technological considerations.

A MATLAB code for phase space reconstruction is listed in Appendix II.A.

The next two sections describe the problem of finding optimal values for m (Section 3.2.1) and τ (Section 3.2.2). Two possible definitions of an optimal embedding are that (1) an embedding is optimal if it delivers the best possible estimates for the topological properties of the attractor and (2) an embedding is optimal if it provides the most accurate forecast of the time series.

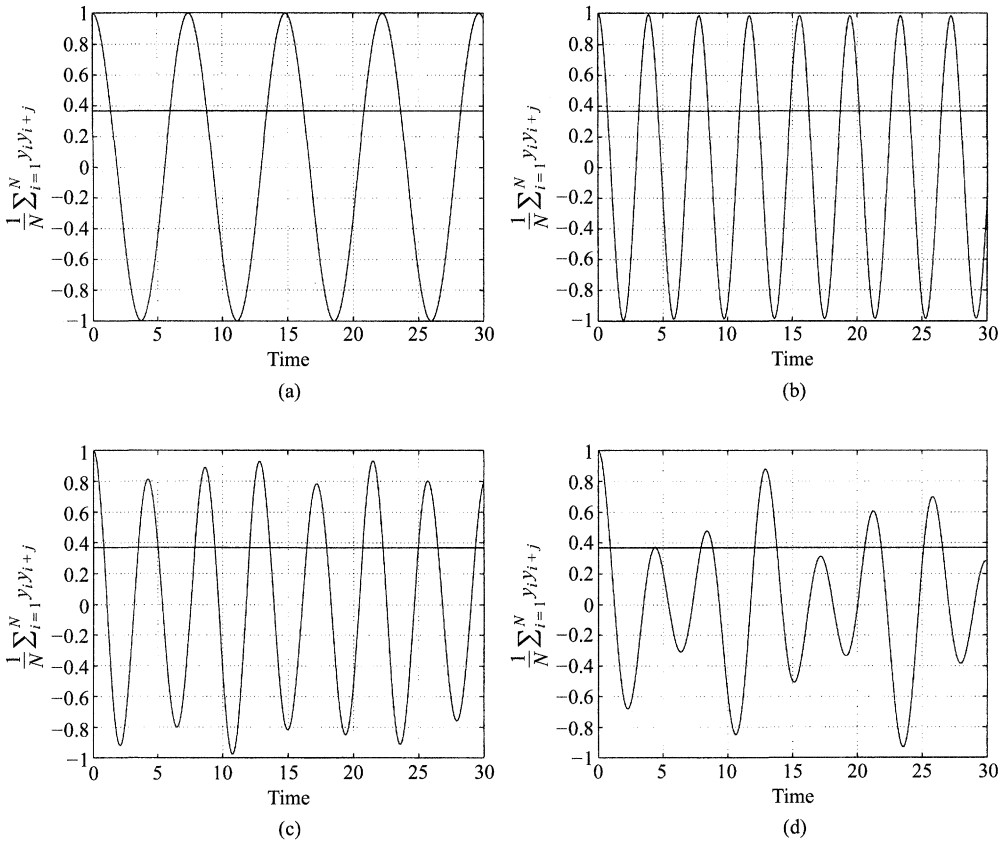


Figure 7 Autocorrelation functions for time series data from the Duffing–Van der Pol oscillator for parameters (a) $\mu = 0.0$, $f = 0.0$; (b) $\mu = 0.2$, $f = 0.0$; (c) $\mu = 0.2, f = 1.0$, $\omega = 0.90$; (d) $\mu = 0.2, f = 1.0$; $\omega = 0.94$. The horizontal line is the value $1/e$.

3.2.1. Optimal Embedding Dimension

Partial answers to the problem of finding an optimal embedding dimension have been provided by Sauer et al. [33]. If the attractor has box-counting dimension D_0, then an embedding dimension of $m \geq 2D_0 + 1$ is sufficient to ensure that the reconstruction is a one-to-one embedding. The one-to-one property is in turn a necessary requirement for forecasting. If the attractor has correlation dimension D_2, then an embedding dimension of $m \geq D_2$ is sufficient to measure the correlation dimension from the embedding. The condition $m \geq D_2$ is also a necessary but not sufficient condition for forecasting (Section 3.3.1). A clear difficulty with these formal bounds is that the fractal dimensions D_0 and D_2 are generally unknown a priori.

In practical applications the Grassberger–Procaccia algorithm can be employed to measure the correlation dimension of reconstructions for different embedding dimensions. The minimum embedding dimension of the attractor is $m + 1$, where m is the embedding dimension above which the measured value of the correlation dimension D_2 remains constant.

An optimal embedding dimension for forecasting can be found in the following utilitarian fashion [34,35]. Forecasts based on the first half of a time series can be constructed for a range of different embedding dimensions. These forecasts can then be compared with the actual time series data from the second half of the time series to find the best forecast (Section 3.3.1) for a given embedding dimension.

3.2.2. Optimal Delay Time

A one-to-one embedding can be obtained for any value of the delay time $\tau > 0$. However, very small delay times will result in near-linear reconstructions with high correlations between consecutive phase space points and very large delays might obscure the deterministic structure linking points along a single degree of freedom. If the delay time is commensurate with a characteristic time in the underlying dynamics, then this too may result in a distorted reconstruction. The optimal delay time for forecasting can be determined using the approach of Holton and May described in the previous section for finding an optimal embedding dimension.

There have been various proposals for choosing an optimal delay time for topological properties based on the behavior of the autocorrelation function. These include the earliest time τ at which the autocorrelation drops to a fraction of its initial value [36] or has a point of inflection [37]. These definitions seek to find times where linear correlations between different points in the time series are negligible, but they do not rule out the possibility of more general correlations.

Fraser and Swinney [38] argue that a better value for τ is the value that corresponds to the first local minimum of the mutual information where the mutual information is a measure of how much information can be predicted about one time series point given full information about the other. Liebert and Schuster [39] have shown that the values of τ at which the mutual information has a local minimum are equivalent to the values of τ at which the logarithm of the correlation sum (Eq. 33) has a local minimum. In seeking a local minimum of $C(r, \tau)$ as a function of τ it is necessary to fix r. Liebert and Schuster suggest employing the smallest value of r where $C(r, \tau)$ scales as r^{-D}.

Some authors have suggested that it is more appropriate to define an optimal embedding window $\tau(m-1)$ rather than optimal values for m and τ separately [40–42].

It is not clear which method if any is superior for all topological properties. However, optimal values based on the behavior of the autocorrelation function are the easiest to compute.

Figure 8a–d show phase space reconstructions using a time delay of $\tau = 1$ for the Duffing–Van der Pol time series with parameters as in Figure 5a–d, respectively. These phase space reconstructions compare favorably with the original phase space portraits in Figure 1a–d. The time delay $\tau = 1$ is optimal in the sense that these are the times at which the autocorrelation function has first decreased to $1/e$ of its original value (see Figure 7). Also shown in Figure 8 are reconstructions of the chaotic time series (Figure 5d) with nonoptimal choices of the time delay. In Figure 8e, $\tau = 0.1$ and the reconstruction is stretched along the diagonal of the embedding, whereas in Figure 8f, $\tau = 20$, and the reconstruction appears to fill a region of the embedding space.

Figure 9 shows a plot of the fractal dimension, measured using the Grassberger–Procaccia algorithm, versus the embedding dimension for the reconstructed phase portrait shown in Figure 8d. The fractal dimension saturates at about $D \approx 2.5$. This

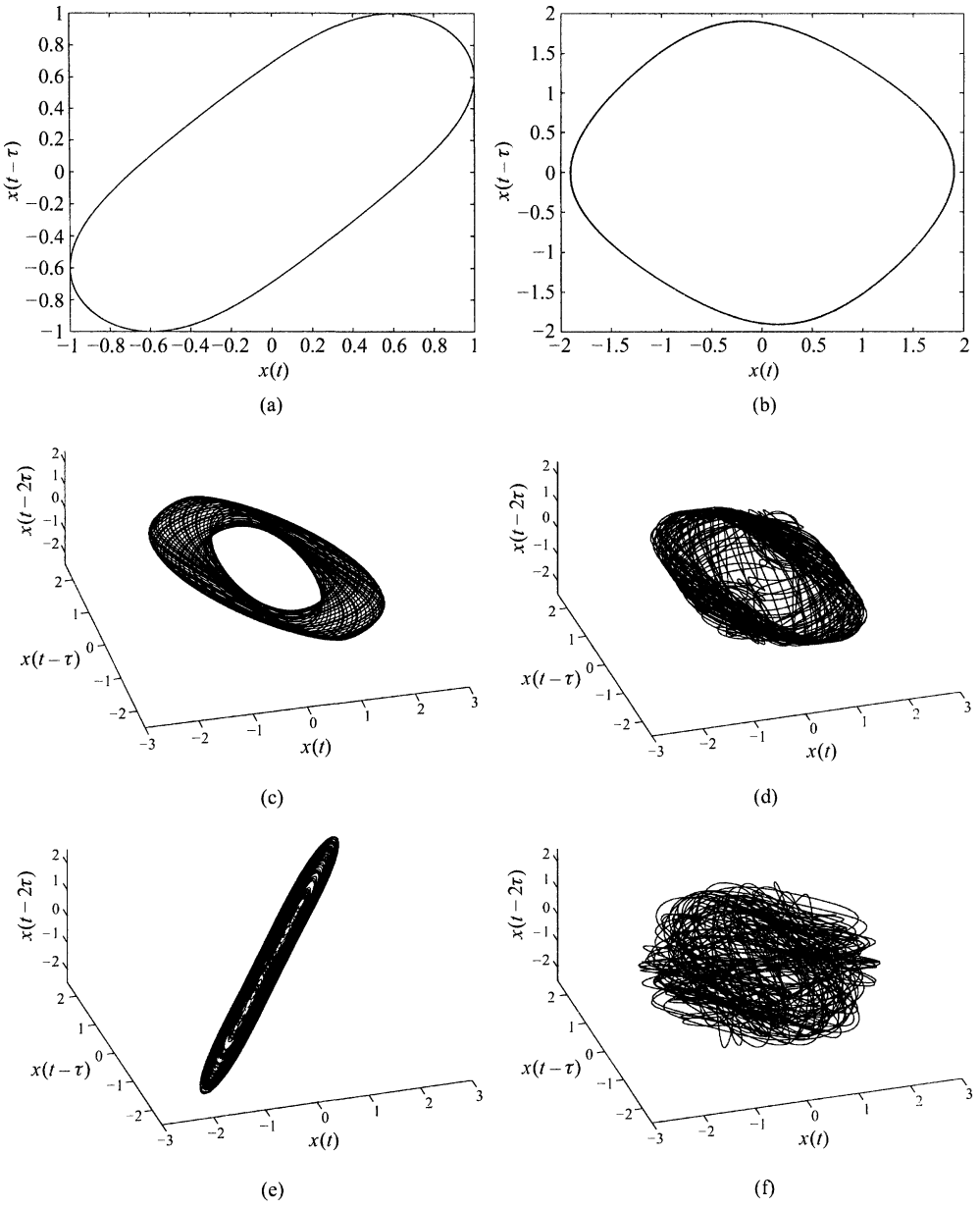


Figure 8 Phase space reconstructions (a)–(d) with embedding dimension three and $\tau = 1$ for time series data from the Duffing–Van der Pol oscillator shown in Figure 5a–d, respectively. Reconstructions are also shown for the time series in Figure 5d with (e) $\tau = .1$ and (f) $\tau = 20$.

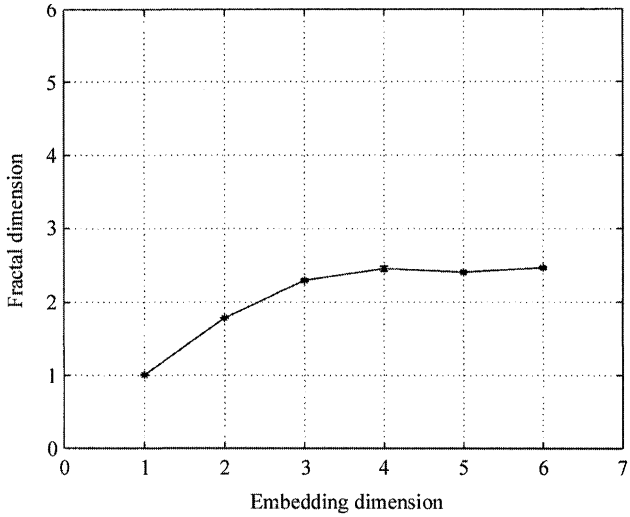


Figure 9 Plot of the fractal dimension versus the embedding dimension for phase space reconstructions of the time series data from the Duffing–Van der Pol oscillator with parameters $\mu = 0.2$, $f = 1.0$, $\omega = 0.94$. The time delay was set to $\tau = 1$ in the reconstructions.

suggests that the minimum embedding for the time series in Figure 5d is three. The value of the fractal dimension based on the time series for just one of the system's dynamical variables is also in good agreement with the measured value of the fractal dimension using all three dynamical variables (Section 1.5).

3.2.3. Measuring Lyapunov Exponents from Time Series

The first algorithms for calculating Lyapunov exponents from a time series were proposed independently by Wolfe et al. [18] and Sano and Sawada [43] in 1985. The first step in these methods is to construct an appropriate embedding of the experimental time series using the method of time delays described in Section 3.2. The maximum Lyapunov exponent can now be calculated as follows. Choose a reference point labeled $\vec{X}(0)$ and the "closest" (see further comments below) neighboring point labeled $\vec{X}^{(1)}(0)$ from the set of reconstructed phase space vectors and calculate

$$\|\Delta \vec{X}_0(0)\| = \|\vec{X}(0) - \vec{X}^{(1)}(0)\| \tag{52}$$

Evolve the two points $\vec{X}(0)$ and $\vec{X}^{(1)}(0)$ forward in the reconstructed phase space for a time T_1 and calculate the new separation distance

$$\|\Delta \vec{X}(T_1)\| = \|\vec{X}(T_1) - \vec{X}^{(1)}(T_1)\| \tag{53}$$

An approximate renormalization is now performed by finding a point $\vec{X}^{(2)}(0)$ that satisfies the dual requirement that (1) $\vec{X}^{(2)}(0)$ is a neighboring point to $\vec{X}(T_1)$ and (2)

$$\Delta \vec{X}_0(T_1) = \vec{X}(T_1) - \vec{X}^{(2)}(0) \tag{54}$$

and $\Delta \vec{X}(T_1)$ are in approximately the same direction. The two points $\vec{X}(T_1)$ and $\vec{X}^{(2)}(0)$ are now evolved for a time T_2 in the reconstructed phase space to calculate

$$\|\Delta \vec{X}(T_1 + T_2)\| = \|\vec{X}(T_1 + T_2) - \vec{X}^{(2)}(T_2)\| \tag{55}$$

The renormalization process of finding a neighboring point to the current point that has a similar orientation to the previous replacement point is repeated N times and then the maximum Lyapunov exponent is calculated as [18]

$$\lambda = \frac{1}{\sum_{k=1}^{N} T_k} \sum_{k=1}^{N} \log \frac{\|\Delta \vec{X}\left(\sum_{j=1}^{k} T_j\right)\|}{\|\Delta \vec{X}_0(T_k)\|} \tag{56}$$

This calculation should then be averaged over several different initial starting points. In implementing the method it is important not to accept as the "closest" neighboring point a point that is temporally separated by a distance less than the delay time τ. This is to avoid choosing adjacent points on the same trajectory. Thus the times $T_1, \ldots, T_N$ should be greater than τ. On the other hand, these times should be small enough to obtain exponential separations.

If the system is ergodic so that the reconstruction phase space attractor is sampled uniformly over time, then the numerical renormalization process can be avoided by following the short-time evolutions of neighboring pairs of points on the attractor and estimating the maximum Lyapunov exponent from [43,44]

$$\lambda(i) = \frac{1}{i\Delta t} \frac{1}{M - i} \sum_{j=1}^{M-i} \log \frac{d_j(i)}{d_j(0)} \tag{57}$$

where $d_j(i)$ is the separation distance between the jth pair of "nearest" neighbors after i discrete time steps.

3.3. Reliability Checks

3.3.1. Forecasting

Holton and May [35] argue that prediction is the sine qua non of determinism and hence the reliability of forecasts should be a fundamental tool for discriminating between deterministic chaos and noise-induced randomness. The following algorithm for predicting time series essentially follows Farmer and Sidorowich [45]. Consider a time series

$$y(\Delta t), y(2\Delta t), \ldots, y(n\Delta t) \tag{58}$$

where Δt is the sampling interval. The aim is to predict $y(n\Delta t + T)$ for some small time T. The first step is to embed the time series to obtain the reconstruction

$$\vec{X}(i\Delta t) = (y(i\Delta t), y(i\Delta t - \tau), \ldots, y(i\Delta t - (m-1)\tau)) \qquad i = n, n-1, \ldots, n-n^* \quad (59)$$

where $n^* = n - (m-1)\tau$ is the number of reconstructed vectors for a time series of length n. Note that the reconstruction slides backward through the data set here. The next step is to measure the separation distance between the vector $\vec{X}(n\Delta t)$ and the other reconstructed vectors and thus to order the reconstructed vectors $\vec{X}^{(1)}, \vec{X}^{(2)}, \ldots \vec{X}^{(n^*-1)}$ so that the separation distance is from smallest to largest i.e.,

$$\|\vec{X} - \vec{X}^{(1)}\| \leq \|\vec{X} - \vec{X}^{(2)}\| \leq \cdots \leq \|\vec{X} - \vec{X}^{(n^*-1)}\| \quad (60)$$

The metric $\|.\|$ is the usual Euclidean metric. Since the $\vec{X}^{(i)}$ are ordered with respect to $\vec{X}(n\Delta t)$, they may be written as $\vec{X}^{(i)}(n\Delta t)$. The next step is to map the $k \geq m+1$ nearest neighbors of $\vec{X}(n\Delta t)$ forward in the reconstructed phase space for a time T. These evolved points are $\vec{X}^{(i)}(n\Delta t + T)$. Suppose that the components of these vectors are as follows:

$$\vec{X}^{(i)}(n\Delta t + T) = \left(x_1^{(i)}(n\Delta t + T), x_2^{(i)}(n\Delta t + T), \ldots, x_m^{(i)}(n\Delta t + T)\right) \quad (61)$$

Now assume a local linear approximation and fit an affine model of the form

$$x_1^{(1)}(n + \Delta t + T) = a_0 + a_1 x_1^{(1)}(n\Delta t) + \cdots + a_m x_m^{(1)}(n\Delta t) \quad (62)$$

$$\vdots$$

$$x_1^{(k)}(n + \Delta t + T) = a_0 + a_1 x_1^{(k)}(n\Delta t) + \cdots + a_m x_m^{(k)}(n\Delta t) \quad (63)$$

The unknown coefficients a_j can be solved using a least-squares method. Finally, the coefficients a_j can be used to construct the prediction

$$y(n\Delta t + T) = x_1(n\Delta t + T) = a_0 + a_1 x_1(n\Delta t) + \cdots + a_m x_m(n\Delta t) \quad (64)$$

The system of equations, Eqs. 62, 63, may be inconsistent or incomplete, in which case the forecast is ignored or k is increased.

Holton and May [35] use the reliability of forecasts to determine optimal values for the delay time and the embedding dimension. The first step in this procedure is to reconstruct the attractor with a delay time τ and an embedding dimension m. Forecasts are then made over times t for N different starting points using the first half of the time series data. The correlation coefficient

$$\rho(\tau, m; t) = \frac{< (x(k,t) - < x(k,t) >)(y(k,t) - < y(k,t) >) >}{\sqrt{< (x(k,t) - < x(k,t) >)^2 >}\sqrt{< (y(k,t) - < y(k,t) >)^2 >}} \quad (65)$$

is then computed as a function of the forecast time t. In Eq. 65, $x(k, t)$ denotes the first component of the evolved vector x in the embedding for the kth forecast after time t, $y(k, t)$ denotes the actual data value corresponding to the kth forecast after time t, and the angular brackets $< >$ denote an average over the k forecasts.

A MATLAB code for forecasting a time series is listed in Appendix II.B.

3.3.2. Surrogate Data

The method of using surrogate data in nonlinear time series analysis was introduced by Theiler et al. in 1992 [46]. This section contains a brief sketch of some of the ideas and methods in that reference. The starting point is to create an ensemble of random nondeterministic surrogate data sets that have the same mean, variance, and power spectrum as the experimental time series data of interest. The measured topological properties of the experimental time series are then compared with the measured topological properties of the surrogate data sets. If both the experimental time series data and the surrogate data sets yield the same values for the topological properties (within the standard deviation measured from the surrogate data sets), then the null hypothesis that the experimental data set is random noise cannot be ruled out.

The method of calculating surrogate data sets with the same mean, variance, and power spectrum but otherwise random is as follows: First construct the Fourier transform of the experimental time series data, then randomize the phases, then take the inverse Fourier transform. An explicit algorithm for achieving this is as follows [47]:

1. Input the experimental time series data $x(t_j), j = 1, \ldots N$ into a complex array

$$z(n) = x(n) + iy(n), \qquad n = 1, \ldots N \tag{66}$$

   where $x(n) = x(t_n)$ and $y(n) = 0$.
2. Construct the discrete Fourier transform

$$Z(m) = X(m) + iY(m) = \frac{1}{N} \sum_{n=1}^{N} z_n e^{-2\pi i(m-1)(n-1)/N} \tag{67}$$

3. Construct a set of random phases

$$\phi_m \in [0, \pi], \qquad m = 2, 3, \ldots \frac{N}{2} \tag{68}$$

4. Apply the randomized phases to the Fourier transformed data

$$Z(m)' = \begin{cases} Z(m) & \text{for } m = 1 \text{ and } m = \frac{N}{2} + 1 \\ |Z(m)|e^{i\phi_m} & \text{for } m = 2, 3, \ldots, \frac{N}{2} \\ |Z(N-m+2)|e^{-i\phi_{N-m+2}} & \text{for } m = \frac{N}{2} + 2,\ \frac{N}{2} + 3, \ldots, N \end{cases} \tag{69}$$

5. Construct the inverse Fourier transform of $Z(m)'$

$$z(n)' = x(n)' + iy(n)' = \frac{1}{N}\sum_{m=1}^{N} Z'_m e^{2\pi i(m-1)(n-1)/N} \tag{70}$$

A MATLAB code for creating surrogate data using the preceding algorithm is listed in Appendix II.C. Figure 10a shows a surrogate data time series sharing the same spectral properties as the time series data for the Duffing–Van der Pol oscillator with parameters as in Figure 5d. A phase space reconstruction for the surrogate data time series using an embedding dimension $m = 3$ and a time delay $\tau = 1$ is shown in Figure 10b.

The phase space portrait for this surrogate data set appears to be more space filling (noisy) than the phase space portrait for the original time series (Figure 8d). Forecasts based on the original time series data and the surrogate time series data using phase space reconstructions with $m = 3$ and $\tau = 1$ in each case are compared in Figure 11a and b, respectively. In Figure 11c the correlation coefficient is computed for 100 forecasts for both the original time series data (solid line only) and the surrogate data (line with crosses). From Figure 11 it is clear that the forecast for the original time series is clearly superior. This is consistent with the apparent randomness in the original time series being due to nonlinear dynamics rather than noise.

4. DISCUSSION

This chapter provided a tutorial-style introduction to the problem of detecting, analyzing and forecasting low-dimensional deterministic chaos in experimental time series. The chapter also contains a set of MATLAB programs for this type of analysis. The detection of deterministic chaos may be considered as a first but very important step

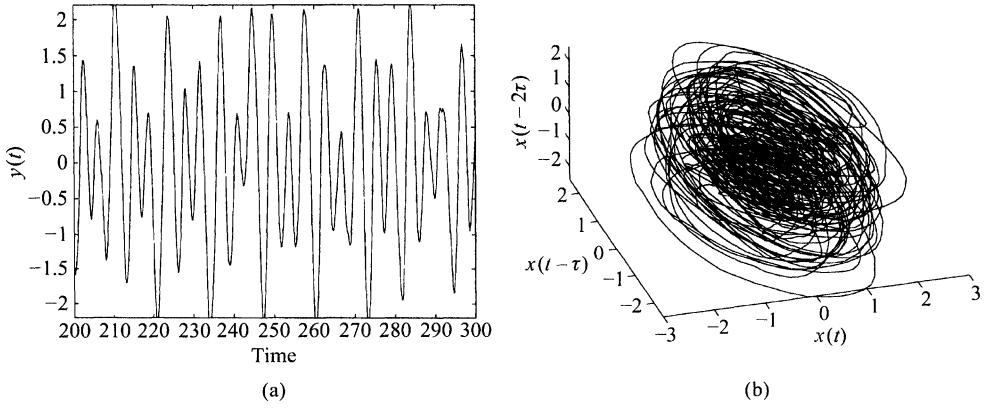


Figure 10 Surrogate data time series (a) and phase space reconstruction (b) with $m = 3$ and $\tau = 1$. The surrogate data have the same spectral properties as the time series data from the Duffing–Van der Pol oscillator for parameters $\mu = 0.2, f = 1.0, \omega = 0.94$.

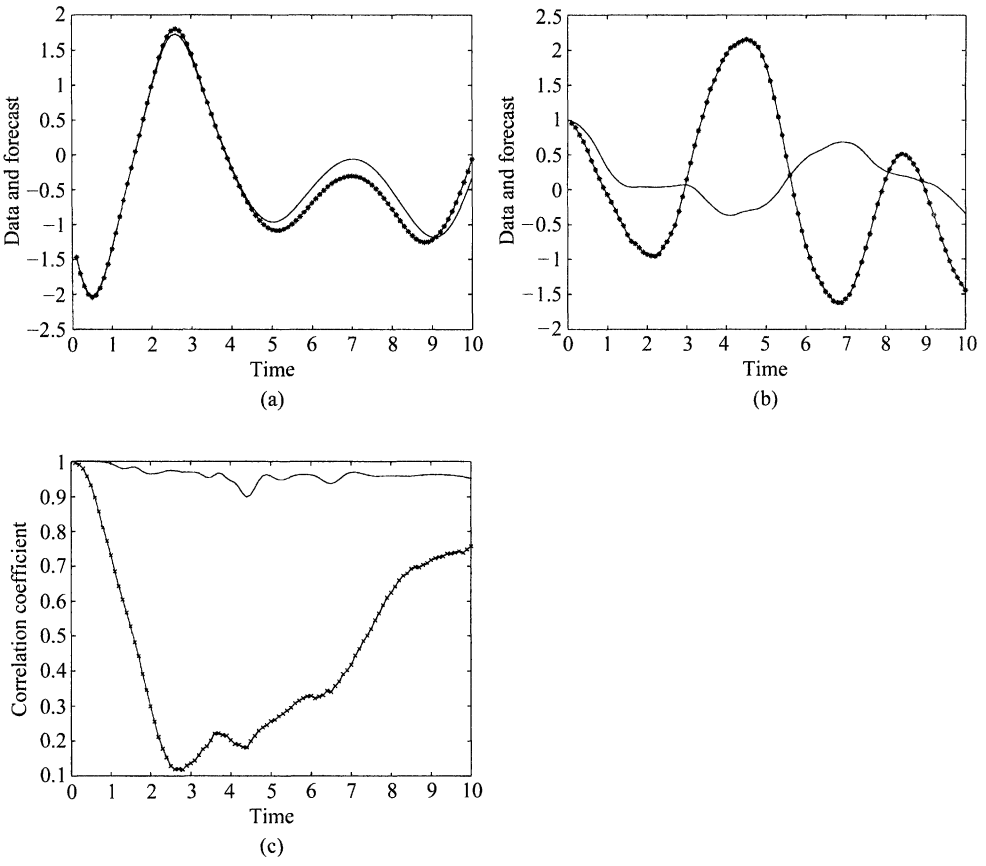


Figure 11 Comparison between forecasts (*) and actual data (solid lines) for (a) time series data from the Duffing–Van der Pol oscillator for parameters $\mu = 0.2, f = 1.0, \omega = 0.94$ and (b) surrogate data time series sharing the same spectral properties. The correlation coefficient based on 100 such forecasts is shown in (c) for the original time series data (solid line only) and the surrogate data (line with crosses).

[48] in a more ambitious program aimed at modeling [49] and ultimately controlling [50] time series data. The methods, algorithms, and computer programs in this chapter constitute a tool box for nonlinear time series analysis in much the same way that standard statistical packages are part of the trade of social scientists. On the other hand, the tool box should not be treated as a black box that can be applied indiscriminately. Some of the questions to address in this process include: Is the time interval between data points in the experimental data set sufficiently small and is the data set sufficiently free from noise to retain deterministic structure if it exists? Has the experimental signal been filtered in any way? Is the time series sufficiently long to permit a reliable reconstruction of the full phase space trajectory? Is the scaling regime in fractal dimension measurements unambiguous? Is the convergence of the fractal dimension of the reconstructed trajectory unambiguous? Is the measured maximum Lyapunov exponent homogeneous over the reconstructed trajectory? If it has a positive value, is this

value significantly different from zero? Can the irregularities in the experimental data be accounted for equally well using linear statistical analysis?

With careful attention to these and other questions, nonlinear time series analysis will provide a valuable adjunct to linear statistical analysis of apparently random-looking biomedical data.

REFERENCES

[1] H. Tong, A personal overview of non-linear time series analysis from a chaos perspective. *Scand. J. Stat.* 22:399–445, 1995.

[2] T. Mullin, A dynamical systems approach to time series analysis. In *The Nature of Chaos*, T. Mullin, (ed.), Chap. 2. Oxford: Oxford University Press, 1993.

[3] V. Isham, Statistical aspects of chaos: a review. In *Networks and Chaos—Statistical and Probabilistic Aspects*, O. E. Barndorff-Nielsen, J. L. Jensen, and W. S. Kendall, (eds.), Chap. 3. London: Chapman & Hall, 1993.

[4] M. Casdagli, Chaos and deterministic versus stochastic non-linear modelling. *J. R. Stat. Soc. B.* 54:303–328, 1992.

[5] N. Gershenfeld, An experimentalist's introduction to the observation of dynamical systems. In *Directions in Chaos*, Vol. 2, Hao Bai-Lin, (ed.), pp. 310–383. Singapore: World Scientific, 1988.

[6] M. R. Guevara, L. Glass, and A. Shrier, Phase locking, period doubling bifurcations, and irregular dynamics in periodically stimulated cardiac cells. *Science* 214:1350–1352, 1981.

[7] D. R. Chialvo, D. C. Michaels, and J. Jalife, Supernormal excitability as a mechanism of chaotic dynamics of activation in cardiac Purkinje fibers. *Circ. Res.* 66:525–545, 1990.

[8] K. M. Stein, N. Lippman, and P. Kligfield, Fractal rhythms of the heart. *J. Electrocardiol.* 24(Suppl.):72–76, 1992.

[9] Z. S. Huang, G. L. Gebber, S. Zhong, and S. Barman, Forced oscillations in sympathetic nerve discharge. *Am. J. Physiol.* 263:R564–R571, 1992.

[10] K. P. Yip, N. H. Holstein-Rathlou, and D. J. Marsh, Chaos in blood flow control in genetic and renovascular hypertensive rats. *Am. J. Physiol.* 261:F400–F408, 1991.

[11] C. D. Wagner, R. Mrowka, B. Nafz, and P. B. Persson, Complexity and "chaos" in blood pressure after baroreceptor denervation of conscious dogs. *Am. J. Physiol.* 269:H1760–H1766, 1996.

[12] C. A. Skarda and W. J. Freeman, How brains make chaos in order to make sense of the world. *Behav. Brain Sci.* 10:161–195, 1987.

[13] D. Hoyer, K. Schmidt, R. Bauer, U. Zwiener, M. Kohler, B. Luthke, and M. Eiselt, Nonlinear analysis of heart rate and respiratory dynamics. *IEEE Eng. Med. Biol. Mag.* 16(1): 31–39, 1997.

[14] W. Szemplinska-Stupnicka and J. Rudowski, Neimark bifurcation, almost periodicity and chaos in the forced van der Pol–Duffing system in the neighbourhood of the principal resonance. *Phys. Lett. A* 192:201–206, 1994.

[15] J. Ford, How random is a coin toss? *Phys. Today* 36(4):40, 1983.

[16] V. I. Oseledec, A multiplicative ergodic theorem. *Trans. Moscow. Math. Soc.* 19:197–231, 1968.

[17] G. Benettin, L. Galgani, and J. M. Strelcyn, Kolmogorov entropy and numerical experiments. *Phys. Rev. A* 14:2338–2345, 1976.

[18] A. Wolfe, J. B. Swift, H. L. Swinney, and J. A. Vastano, Determining Lyapunov exponents from a time series. *Physica D* 16:285–317, 1985.

[19] H. Fujisaka, Multiperiodic flows, chaos and Lyapunov exponents. *Prog. Theor. Phys.* 68:1105–1119, 1982.

[20] M. Casartelli, E. Diana, L. Galgani, and A. Scotti, Numerical computations on a stochastic parameter related to Kolmogorov entropy. *Phys. Rev. A* 13:1921–1925, 1976.
[21] I. Shimada and T. Nagashima, A numerical approach to ergodic problem of dissipative systems. *Prog. Theor. Phys.* 61:1605–1613, 1979.
[22] G. Benettin, L. Galgani, A. Giorgilli, and J.-M. Strelcyn, Lyapunov characteristic exponents for smooth dynamical systems: A method for computing all of them. *Meccanica* 15:9–19, 1980.
[23] B. Mandelbrot, *The Fractal Geometry of Nature*. New York: Freeman, 1982.
[24] H. G. E. Hentschel and I. Procaccia, The infinite number of generalized dimensions of fractals and strange attractors. *Physica D* 8:435–444, 1983.
[25] J. Kaplan and J. Yorke, Functional differential equations and the approximation of fixed points. In *Lecture Notes in Mathematics*, No. 730. Berlin: Springer-Verlag, 1979.
[26] P. Grassberger and I. Procaccia, Characterization of strange attractors. *Phys. Rev. Lett.* 50:346–349, 1983.
[27] D. Ruelle, Deterministic chaos: The science and the fiction. *Proc. R. Soc. Lond. A* 427:241–248, 1990.
[28] C. Essex and M. A. H. Nerenberg, Comments on "Deterministic chaos: the science and the fiction" by D. Ruelle. *Proc. R. Soc. Lond. A* 435:287–292, 1991.
[29] S.-Z. Hong and S.-M. Hong, An amendment to the fundamental limit on dimension calculations. *Fractals* 2:123–125, 1994.
[30] N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw, Geometry from a time series. *Phys. Rev. Lett.* 45:712–716, 1980.
[31] F. Takens, Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence*, D. A. Rand and L. S. Young, eds. Berlin: Springer, 1981.
[32] T. Sauer and J. A. Yorke, Rigorous verification of trajectories for computer simulation of dynamical systems. *Nonlinearity* 4:961–979, 1991.
[33] T. Sauer, J. Yorke, and M. Casdagli, Embedology. *J. Stat. Phys.* 65:579–616, 1994.
[34] G. Sugihara and R. M. May, Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series. *Nature* 344:734–741, 1990.
[35] D. Holton and R. M. May, Distinguishing chaos from noise. In *The Nature of Chaos*, Chap. 7. Oxford: Oxford University Press, 1993.
[36] A. M. Albano, J. Muench, C. Schwartz, A. I. Mees, and P. E. Rapp, Singular-value decomposition and the Grassberger–Procaccia algorithm. *Phys. Rev. A* 38:3017, 1988.
[37] G. P. King, R. Jones, and D. S. Broomhead, Phase portraits from a time series: A singular system approach. *Nucl. Phys. B* 2:379, 1987.
[38] A. M. Fraser and H. Swinney, Independent coordinates for strange attractors from mutual information. *Phys. Rev. A* 33:1134–1139, 1986.
[39] W. Liebert and H. G. Schuster, Proper choice of the time delay for the analysis of chaotic time series. *Phys. Lett. A* 142:107, 1989.
[40] A. M. Albano, A. Passamante, and M. E. Farrell, Using higher-order correlations to define an embedding window. *Physica D* 54:85, 1991.
[41] J. M. Martinerie, A. M. Albano, A. I. Mees, and P. E. Rapp, Mutual information, strange attractors, and the optimal estimation of dimension. *Phys. Rev. A.* 45:7058, 1992.
[42] M. T. Rosenstein, J. J. Collins, and C. J. de Luca, Reconstruction expansion as a geometry-based framework for choosing proper delay times. *Physica D* 73:82–98, 1994.
[43] M. Sano and Y. Sawada, Measurement of the Lyapunov spectrum from a chaotic time series. *Phys. Rev. Lett.* 55:1082–1085, 1985.
[44] M. T. Rosenstein, J. J. Collins, and C. J. de Luca, A practical method for calculating largest Lyapunov exponents from small data sets. *Physica D* 65:117–134, 1994.
[45] J. D. Farmer and J. J. Sidorowich, Predicting chaotic times series, *Phys. Rev. Lett.* 59: 845–848, 1987.

[46] J. Theiler, B. Galdrikian, A. Longtin, S. Eubank, and J. D. Farmer, Using surrogate data to detect nonlinearity in time series. In *Nonlinear Modeling and Forecasting*, M. Casdagli and S. Eubank, eds. New York: Addison-Wesley, 1992.

[47] J. Mazaraki, Dynamical methods for analysing and forecasting chaotic data. Honours thesis, Applied Mathematics, University of New South Wales, 1997.

[48] D. Ruelle, Where can one hope to profitably apply the ideas of chaos? *Phys. Today* July: 24–30, 1994.

[49] B. R. Noack, F. Ohle, and H. Eckelmann, Construction and analysis of differential equations from experimental time series of oscillatory systems. *Physica D* 56:389–405, 1992.

[50] E. Ott and M. Spano, Controlling chaos. *Phys. Today* May: 34–40, 1995.

APPENDIX

I. Dynamical Systems Analysis—MATLAB Programs

A. Numerical Integration of Three Coupled ODEs

```
% File: VanDerPolSolv.m
%
% Numerical integration of Duffing-Van Der Pol Oscillator.
% Ordinary differential equation solver using in-built
% Matlab function (ode45).
%
% Uses: VanDerPol.m
% Set initial conditions for the three differential equations
x0=[1;0;0];
% Set integration time
timePoints=5000:0.3:7500;
% Solve equations
options=odeset('RelTol',1e-10,'AbsTol',1e-12);
[t,x]=ode45('VanDerPol',timePoints,x0);

function xp-VanDerPol(t,x)
%
% Duffing-Van Der Pol equations expressed as a first order system
% (see equations 3-5). Parameter values (mu, f and omega)
% can be chosen to demonstrate various trajectories (see text).
%
% IN:
% t: time (not used, but necessary for ODE solver)
% x: Input vector
%
% OUT:
% xp: dx/dt
% Current parameters demonstrate chaotic behavior of the oscillator.
mu=0.2; f=1.0; omega=0.94;
% define equations
xp(1)=x(2);
xp(2)=mu*(1-x(1)^2)*x(2)-x(1)^3+f*cos(x(3));
xp(3)=omega;
% transpose into column vector for ODE solver
xp=xp';
```

B. Three-Dimensional Phase Space Plots

```
% File: PSplot.m
%
% Three-dimensional phase space plot of three coupled ODEs
% (data assumed to be stored in matrix 'x' from VanDerPolSolve.m).
%
% Transform x3 to sin(x3)
xnew=x;
xnew(:,3)=sin(x(:,3));
% Generate 3D plot
plot3(xnew(:,1),xnew(:,2),xnew(:,3));
xlabel('x1');
ylabel('x2');
zlabel('sin(x3)')
title('Duffing-Van der Pol oscillator.');
rotate3d on
view([-5,58]);
```

C. Two-Dimensional Surface of Section

```
% File: SurfaceOfSection.m
%
% Two dimension Poincare surface of section on plane x3=0
% (data assumed to be stored in matrix 'xnew' generated from PSplot.m).
%
% Start with empty surface
clear Surface
OldDistance=0;
k=1;
for i=1:size(xnew)*[1;0]
     NewDistance=xnew(i,3);
     if (NewDistance>=0 * OldDistance<0)
     % Add new point to the surface
     TotalDistance=NewDistance-OldDistance;
     Surface(k,:)=xnew(i-1,:)-(OldDistance/Total Distance)*...
        (xnew(i,:)-xnew(i-1,:));
     k=k+1;
   end
   OldDistance=NewDistance;
end
% Generate 2D plot
plot(Surface(:,1),Surface(:,2),'*');
xlabel('x1');
ylabel('x2');
title('Poincare Surface of Section.');
```

D. Lyapunov Exponents for Three Coupled ODEs

```
% File: LyapunovSolver.m
%
% Calculate all the Lyapunov exponents for the Duffing-Van der Pol oscillator.
%
% Uses: IntegrateVDPSystem.m, GramSchmidt.m
% Step 1: Construct a unit hypersphere (Dimension=Dim; axis at time 0:
% a(1)=[1;0;0...;0], a(2)=[0;1;0...;0],...,
a(n)=[0;0;0...;1]
% centered on an initial point (x(1,:))
% n=number of iterations to average
n=5000;
% tau
tau=0.1;
Sigma=zeros(n,3);
% Initial conditions
x0=[1;0;0];
a0=eye(3);
x=zeros(3,n);
% j=1 (Iteration variable)
for j=1:n
   % Step2: Integrate the nonlinear equation of motion over a
   % characteristic time scale tau to obtain x(j*tau).
   % Integrate the variational equations to obtain the evolved
   % axis vectors (a1,a2,...,an) for time tau.
   [xpoint,a]=IntegrateVDPSystem(x0,a0,tau);
   % x: numerical solution of the system {NumberOfPoints.Dim}
   x(:,j)=xpoint;
   % Step3: Apply Gram-Schmidt re-orthonormalization to the axis vectors.
   [aUnit]=GramSchmidt(a);
   %Step4: Compute Sigma where
   % Sigma(j,1)=log(norm(a(1)),
   Sigma(j,2)=log(norm(a(2)),...,
```

```
   % Sigma(j,n)=log(norm(a(n))
   for i=1:3
     aOrth(:,i)=dot(a(:,i),aUnit(:,i))*aUnit(:,i);
   end
   for i=1:3
     Sigma(j,i)=log(norm(aOrth(:,i),2));
   end
   x0=xpoint;
   a0=aUnit;
end
% Step 5: Compute the Lyapunov exponents
Lyapunov=(1/(n*tau))*sum(Sigma,1);

function [x,a]=IntegrateVDPSystem(x0,a0,tau)
%
% This function integrates the Duffing-Van der Pol equations
% and the variational equations returning the final result at time tau
% for initial conditions x0 and a0 (3 vectors associated with variational
% equations).
%
% IN:
% x0: Vector {3} of initial conditions
% a0: Matrix {3,3} with initial vectors associated with variational equations.
% tau: Final time.
%
% OUT:
% x: Vector {3} with solution at time tau
% a: Matrix {3,3} with solution of 3 vectors at time tau.
%
% Uses: VanDerPolVarEqu.m
s0=zeros(12,1);
s0=x0;
for i=1:3
  s0(i*3+1:i*3+3)=a0(:,i);
end
t0=0
tfinal=t0+tau;
options=odeset('RelTol', 1e-10,'AbsTol',1e-12);
[t,s]=ode45('VanDerPolVarEqu',[t0 tfinal], s0);
DataLength=size(s)*[1;0];
x=s(DataLength,1:3)';
for i=1:3
  a(:,i)=s(DataLength,i*3+1:i*3+3)';
end

function sp=VanDerPolVarEqu(t,s)
%
% Duffing-Van Der Pol equations expressed as a first order system
% (see equations 3-5). Parameter values (mu, f and omega)
% can be chosen to demonstrate various trajectories (see text).
% Includes Variational Equations necessary to solve Jacobian sp(4:12).
%
% Current parameters demonstrate chaotic behavior of the oscillator.
%
% IN:
% t: time (not used, but necessary for ODE solver)
% s: Input Vector
%
% OUT:
% sp: ds/dt
%
% Dimension of the system
n=3;
% Initial parameters describing chaotic behavior
mu=0.2; f=0.0; omega=1.0;
```

```
% define equations
sp(1)=s(2);
sp(2)=mu*(1-s(1)^2)*s(2)-s(1)^3+f*cos(s(3));
sp(3)=omega;
for i=1:n
  sp(n*i+1)=s(n*i+2);
  sp(n*i+2)=(-2*mu*s(1)*s(2)-3*s(1)^2)*s(n*i+1)+...
    mu*(1-s(1)^2)*s(n*i+2)+f*s(n*i+3);
  sp(n*i+3)=0;
end
% transpose into column vector for ODE solver
sp=sp';
```

```
function [ReNormMatrix]=GramSchmidt(Matrix)
%
% Compute a set of normal-orthogonal vectors using the Gram-Schmidt algorithm.
% Matrix contains vector Matrix(:,1),Matrix(:,2),..., Matrix(:,3)
%
% IN:
% Matrix: Matrix containing original vectors.
%
% OUT:
% ReNormMatrix: Matrix with the renormalized set of vectors
Dim=size(Matrix)*[1;0];
ReNormMatrix=zeros(size(Matrix));
ReNormMatrix(:,1)=Matrix(:,1)/norm(Matrix(:,1),2);
for j=2:Dim
  z=Matrix(:,j);
  for i=j-1:-1:1
    z=z-dot(Matrix(:,j),ReNormMatrix(:,i))*ReNormMatrix(:,i);
  end
  ReNormMatrix(:,j)=z/norm(z,2);
end
```

E. Grassberger–Procaccia Algorithm

```
% File: CorrDim.m
%
% Implements the Grassberger-Procaccia algorithm to measure the correlation
% dimension (data assumed to be stored in matrix 'x' from VanDerPolSolve.m).
%
% Uses: Distance.m, BinFilling.m, Slope.m
% Transform x3 to sin(x3)
xnew=x;
xnew(:,3)=sin(x(:,3));
% Number of valid points in xnew matrix.
NoPoints=[size(xnew)*[1;0] size(xnew)*[1;0] size(xnew)*[1;0]];
% Calculates 32 equi-spaced bins on a logarithmic scale
[Radius]=Distance(xnew, NoPoints);
% Fills bins with the number of pairs of points with separation given by Radius.
[BinCount]=BinFilling(xnew,NoPoints,Radius);
MaxEmDim=3;
% Normalizes the matrix Radius, by its maximum value
% (for each dimension independently).
for i=1:MaxEmDim
  max=Radius(32,i);
  RadiusNormal(:,i)=Radius(:,i)/max;
end
% Plots the BinCount for specific Radius for all Embedding Dimensions.
figure(1);
for i-1:MaxEmDim
  if i==1
      hold off
  end
  loglog(RadiusNormal(:,i),BinCount(:,i),'+-');
```

```
   if i==1
      hold on
   end
end
% Plot correlation Integral
ok=1;
title(strcat('Correlation Integral'));
xlabel(strcat('% of Maximum Radius, MaxEmDim=',num2str (MaxEmDim),')'));
ylabel('% of Number of Points');
% Find the slope for all the dimensions.
for i=1:MaxEmDim
   [Slopes(i), SlopeErrors(i)]=Slope(Radius(:,i),
   BinCount(:,i), .6 .125);
end
% Plot Fractal Dimensions
figure(2)
Dim=1:MaxEmDim;
errorbar(Dim,Slopes,SlopeErrors, 'b*-');
axis([0 MaxEmDim+1 0 MaxEmDim]);
grid on
zoom on;
title(strcat('Data Set: ','Duffing VanDerPol Oscillator'));
xlabel('Dimension');
ylabel('Fractal Dimension');

function [Radius]=Distance(Portrait, NoPoints);
%
% IN:
% Portrait: Is the matrix {Number of data points, MaxEmDim} in which
%    the trajectories are contained.
% NoPoints: Is the vector {1,MaxEmDim} in which the number of valid
%    points for each dimension is contained.
%
% OUT:
% Radius: Is the matrix {32,MaxEmDim} in which the difference between
%    the maximum and minimum distances from one point to any other, is divided
%    into 32 logarithmically equal intervals (for all dimensions).
%
% Uses: DistVectPoint.m, ElimZero.m
MaxEmDim=size(Portrait)*[0;1];
NoPointsEnd=[NoPoints 1];
MinDist=ones(1,MaxEmDim)*1e20;
MaxDist=zeros(1,MaxEmDim);
Radius=zeros(32,MaxEmDim);
for EmDim=1:MaxEmDim
   minval=zeros(1,EmDim);
   minloc=zeros(1,EmDim);
   maxval=zeros(1,EmDim);
   maxloc=zeros(1,EmDim);
   for i=NoPointsEnd(EmDim):-1:NoPointsEnd(EmDim+1)+1
      % Calculates the distances from point Portrait(i,1: EmDim) to all the
      % points in matrix Portrait (1:i-1,1:EmDim)
      Distances=DistVectPoint(Portrait(1:1-1,1:EmDim), Portrait(i,1:EmDim));
      % Eliminates all points with distance less than Tolerance=1e-10
      DistanceNoZero=ElimZero(Distances, 1e-10);
      [minval,minloc]=min(DistanceNoZero,[],1);
      [maxval,maxloc]=max(Distances,[],1);
      for j=1:EmDim;
         if MinDist(j)>minval(j)
              MinDist(j)=minval(j);
         end
         if MaxDist(j)<maxval(j)
              MaxDist(j)=maxval(j);
         end
      end
```

```
  end
end
% Fill 32 bins (equally spaced logarithmically)
for k-1:32
  Radius(k,:)exp(log(MinDist)+k*(log(MaxDist)-log(MinDist))/32);
end

function [Distance]=DistVectPoint(data,point);
%
% This function calculates the distance from all elements of the matrix
% {n,MaxDim} to the point {1,MaxDim}, for all the dimensions.
%
% IN:
% data: Is a matrix {n,MaxDim} of n points to which 'point' is going
    to be compared.
% point: Is a vector {1,MaxDim} that represents the 'point' in MaxDim
%   dimensions.
%
% OUT:
% Distance: Is a matrix {n,MaxDim} that contains the distances from
%   'point' to all other points in 'data' (for dimensions 1 to MaxDim).
%
% Example:  data=[      0          0          0
%                       0          0          1
%                       0          1          1
%                       1          0          1
%                       1          1          1 ];
%             point=[   0          0          0 ];
%         Distance=[    0          0          0
%                       0          0          1.0000
%                       0          1.0000     1.4142
%                       1.0000     1.0000     1.4142
%                       1.0000     1.4142     1.7321 ]
Diffe=zeros(size(data));
for i=1:size(data)*[0;1]
  Diffe(:,i)=data(:,i)-point(i);
end
% Calculate Euclidean distance
Diffe=Diffe.^2;
Distance=cumsum(Diffe,2);
Distance=sqrt(Distance);

function DistanceNoZero=ElimZero(Distance, Tolerance);
%
% Replaces all points with distance less than Tolerance with 1e20.
% This is necessary in the bin-filling algorithm.
%
% IN:
% Distance: Is a matrix {n,MaxDim} that contains the distances from
%   'point' to all other points in 'data' (for dimensions 1 to MaxDim).
% Tolerance: Is a scalar that determines the minimum distance to be
considered.
%
% OUT:
% DistanceNoZero: Is a matrix {n,MaxDim} equal to Distance, but with all
%   elements smaller than Tolerance replaced with 1e20
SigDist=Distance-Tolerance;
SigDist=((sign(sign(SigDist.*-1)-0.5))+1)*1e20;
DistanceNoZero=Distance+SigDist;

function [BinCount]=BinFilling(Portrait,NoPoints,Radius)
%
% IN:
% Portrait: Is the matrix {Number of data points, MaxEmDim} in which the
%   trajectories are contained.
```

```
% NoPoints: Is the vector {1,MaxEmDim} in which the number of points for each
%    dimension is contained.
% Radius: Is the matrix {32,MaxEmDim} in which the difference between the
%    maximum and minimum distances from one point to any other, is divided
into
%    32 logarithmically equal intervals (for all dimensions)
%
% OUT:
% BinCount: Is a matrix {32,MaxEmDim} with the total count of pair of points
%    with a distance smaller than that specified by Radius for the 32
intervals.
%
% Uses: DistVectPoint.m, CountPoints.m
MaxEmDim=size(Portrait)*[0;1];
BinCount=zeros(32,MaxEmDim);
NoPointsEnd=[NoPoints 1];
for EmDim=1:MaxEmDim
     for i=NoPointsEnd(EmDim):-1:NoPointsEnd(EmDim+1)+1
       Distances=zeros(i-1,EmDim);
       Distances=DistVectPoint(Portrait(1:i-1,1:EmDim),
       Portrait(i,1:EmDim));
       for j=1:32
         BinCount(j,1:EmDim)=BinCount(j,1:EmDim)+...
           CountPoints(Distances,Radius(j,1:EmDim));
       end
     end
end
BinCount=BinCount./(((ones(32,1)*NoPoints).*(ones(32,1)*NoPoints-1))/2);

function [CountVect]=CountPoints(Distances,Threshold);
%
% IN:
% Distance: Is a matrix {n,MaxDim} that contains the distances from 'point'
%    to all other points in 'data' for dimensions 1 to MaxDim.
% Threshold: Is the upper bound on Distance.
%
% OUT:
% CountVect: Is a vector {1,MaxDim] with the count of distances smaller
%    than Threshold
VectLength=length(Threshold);
NumOfPoints=size(Distances)*[1;0];
CountVect=zeros(1,VectLength);
ThresholdMatr=ones(NumOfPoints,1)*Threshold;
CountVect=sum((Distances<ThresholdMatr),1);

function [Slope, SlopeError]=Slope(RadiusV, BinCountV, center, high)
%
% This function gives the slope and error for a line (in a logarithmic scale)
% given by the points RadiusV, BinCountV. The only relevant points are the ones
% that are contained in the band center-high/2, center+high/2.
%
% The values for center define the position of the center of the band and can
% range from 0 to 1 with 1 at the top.
%
% IN:
% RadiusV: Vector with the radii limits of a specific Dimension.
% BinCountV: Vector containing the counts of pairs of elements with distance
%    smaller than radius.
% Center: Center position where the slope is to be evaluated.
% High: Band size around center.
%
% OUT:
% Slope: Slope evaluated in the region center-high/2, center+high/2.
% SlopeError: Error of the evaluated fitted line to the original data.
lnRadiusV=log(RadiusV);
lnBinCountV=log(BinCountV);
```

```
Max=0;
Min=lnBinCountV(1);
IntervalHigh=(Max-Min)*high;
Top=-((Max-Min)*(1-center))+(IntervalHigh/2);
Base=-((Max-Min)*(1-center))-(IntervalHigh/2);
k=1;
for i=1:32
     if((lnBinCountV(i)>=Base & lnBinCountV(i)<=Top))
          RelDataX(k)=lnRadiusV(i);
          RelDataY(k)=lnBinCountV(i);
          k=k+1;
     end
end
[P,S]=polyfit(RelDataX,RelDataY,1);
Slope=P(1);
SlopeError=S.normr;
```

II. Time Series Analysis—MATLAB Programs

A. Phase Space Reconstruction

```
function [Portrait, NoPoints, MaxPosEmDim]=...
     Trajectory(SigData, MaxEmDim, TimeDelay)
%
% This function creates a matrix containing the MaxEmDim trajectories generated
% for a specified time delay and a given set of data (SigData).
%
% IN:
% SigData: Vector (of the form n,1) to be analyzed.
% MaxEmDim: The maximum embedding dimension for which the trajectory
(portrait)
%    is to be constructed.
% TimeDelay: Time delay in number of points.
%
% OUT:
% Portrait: Matrix in which each row is a point in the reconstructed trajectory.
%    Each point in the row is the coordinate of that point.
% NoPoints: Number of points for each dimension. For any dimension EmDim,
%    NoPoints=length(SigData)-(EmDim-1)*TimeDelay.
% MaxPosEmDim: Maximum possible embedding dimension for the number of
points in
%    SigData.
DataLength=length(SigData);
MaxPosEmDim=floor(2*log10(DataLength));
clear NoPoints
for i=1:MaxEmDim
     NoPoints(i)=DataLength-((i-1)*TimeDelay);
end
clear Portrait;
Portrait=zeros(NoPoints(1),MaxEmDim);
for i=1:MaxEmDim
     Portrait(1:DataLength-((i-1)*TimeDelay),i)=...
       SigData(((i-1)*TimeDelay)+1:DataLength);
end
```

B. Forecasting

```
function [yFuture]=Forecast(y,NoPointsForcast,TimeDelay,EmDim,Redund)
%
% IN:
% y: Vector with original data.
% NoPointsForcast: Number of points to forecast.
% TimeDelay: Time delay for the trajectory reconstruction.
% EmDim: Dimension for the trajectory reconstruction.
% Redund: Number of redundant points to evaluate forecasting parameters.
%
% OUT:
% yFuture: Vector {NoPointsForcast} with forecast data.
%
% Uses: BackEmbedding.m, ClosestPoints.m, EvalFuture.m,
yFuture=zeros(NoPointsForcast,1);
% Backward embedding
[x,NoPoints]=BackEmbedding(y,TimeDelay,EmDim);
% Find closest points
x0=x(NoPoints,:);
% EmDim+1 is the minimum to solve the problem.
% Can be increased to evaluate more points.
```

```
k=(EmDim+1)+Redund;
[xClosest,PosClosest]=ClosestPoints(x(1:NoPoints-NoPointsForcast,:),x0,k);
for i=1:NoPointsForcast
[xClosestFuture]=EvalFuture (x,PosClosest,i,k);
     % Calculate the set of parameters 'a' that best generate the forecast of
     % xClosestFuture from xClosest.
     k=size(xClosest)*[1;0];
     a=regress(xClosestFuture(:,1),[ones(k,1),xClosest]);
     % Forecast y
     yFuture(i,:)=[1, x0]*a;
end

function [xClosest, PosClosest]=ClosestPoints(x,x0,N)
%
% Searches for the N closest points to x0 in matrix x.
%
% IN:
% x: Matrix {of the form n,EmDim} with n points of dimension EmDim.
% x0: Vector {of the form 1, EmDim} to where the closest points
%    in x are search.
% N: Number of points to look for.
%
% OUT:
% xClosest: Matrix {of the form N, EmDim} with the N closest points
%    of x to x0.
% PosClosest: Vector {of the form N,1} with the position of the
%    closest points.
[NoPoints, EmDim]=size(x);
Distance=sum(((x-ones(NoPoints,1)*x0).^2),2);
[Distance, I]=sort(Distance);
xClosest=zeros(N,EmDim);
PosClosest=zeros(N,1);
for i=1:N
     xClosest(i,:)=x(I(i),:);
end
PosClosest=I(1:N);

function [xClosestFuture]=EvalFuture(x,PosClosest,NoPoints Forcast,N)
%
% Evaluate the trajectory (x) for each point indicated by the vector
% PosClosest N steps in the future.
%
% IN:
% x: Matrix {of the form n,EmDim} with n points of dimension
%    EmDim.
% PosClosest: Vector {of the form N,1} with the position of the
%    closest points.
% NoPointsForcast:
% N: Number of points.
%
% OUT:
% xClosestFuture: Matrix {of the form N, EmDim} with the evolved
%    points in matrix PosClosest, NoPoints ahead in the future.
[NoPoints,EmDim]=size(x);
xClosestFuture=zeros(length(PosClosest),EmDim);
for i=1:N
     xClosestFuture(i,:)=x(PosClosest(i)+NoPointsForcast,:);
end
```

C. Surrogate Data

```
function [VectorOut]=SurrDataFFT(VectorIn)
%
% This function assigns random phases for all frequencies of the input
% vector (VectorIn{n,1}) in its Fourier representation.
%
% IN:
% VectorIn: Vector {of the form n,1} with original data.
%
% OUT:
% VectorOut: Vector {of the form n,1} with surrogate data.
VectorLength=length(VectorIn);
% FFT of original Signal
Vectorfft=fft(VectorIn);
% Randomize Phase
NRel=ceil((VectorLength-1)/2)+1;
NChange=VectorLength-NRel;
RelVector=zeros(NRel,1);
RelVector=Vectorfft(1:NRel);
RandAngles=rand(NChange,1)*2*pi;
RelVector(2:NChange+1)=(cos(RandAngles)+sin(RandAngles)*i).*...
     abs(Vectorfft(2:Nchange+1));
VectorRandom=zeros(VectorLength,1);
NRel=ceil((VectorLength-1)/2)+1;
VectorRandom(1:NRel)=RelVector;
for i=VectorLength:-1:NRel+1
     j=VectorLength-i+2;
     VectorRandom(i)=conj(RelVector(j));
end
% IFFT to generate new signal
VectorOut=real(ifft(VectorRandom));
% VectorOut: Vector {of the form n,1} with surrogate data.
```