Tema 14

Sistemas dinámicos

14.1. Introducción

En física se utiliza el término *sistema dinámico* para referirse a cualquier tipo de modelo matemático utilizado para describir la evolución temporal de un proceso. La forma de caracterizar matemáticamente ese proceso es definiendo unas variables que representan los observables del proceso (por ejemplo, la población de individuos de una determinada especie, la concentración química de un elemento, la energía, la temperatura, etc.) y las leyes de su evolución, es decir, los mecanismos por los que estas cantidades cambian en el tiempo. Si la evolución temporal ocurre a través de una serie discreta de tiempos separados por intervalos finitos se dice que el sistema dinámico es discreto. Si el tiempo varía de forma continua de modo que podamos obtener el valor de esos observables en cualquier instante de tiempo entonces el sistema dinámica es continuo.

Los sistemas dinámicos pueden ser deterministas o estocásticos. En el primer caso, la evolución temporal de las variables del sistema está completamente determinada por la condición inicial, esto es, si partimos desde el mismo estado inicial siempre obtendremos la misma evolución temporal. Un ejemplo de sistema dinámico determinista son los autómatas celulares que veremos en el próximo capítulo. En el caso de los sistemas estocásticos tenemos términos aleatorios que hacen que una misma condición inicial dé origen a diferentes resultados. Un ejemplo de sistema dinámico estocástico son los caminantes aleatorios que vimos en el capítulo 12. En este capítulo sólo trataremos con sistemas dinámicos deterministas.

El *espacio de fases* de un sistema dinámico está compuesto por el conjunto de valores que pueden tomar las variables del sistema. Las dimensiones de ese espacio vendrán dadas por el número de variables del sistema dinámico. Supongamos por ejemplo que estamos considerando un modelo depredador-presa con dos variables: X será el número de presas (conejos) e Y el número de depredadores (zorros). Ambas especies se reproducen, lo que conlleva un aumento de ambas poblaciones, pero también existe una mortalidad y lo que es más importante, existe una interacción entre ellas ya que los depredadores se alimentan de las presas. Por ejemplo, una población alta de zorros provocará un descenso en la población de conejos, lo que a la postre repercutirá en la población de zorros, mientras que una población alta de conejos favorecerá el crecimiento de la población de zorros y esto tendrá un efecto negativo sobre el número de conejos. El espacio de fases estará compuesto por todos los valores que pueden tomar la población de depredadores y de presas. Podemos entonces repre-

sentar ese espacio de fases en un plano, en el que el eje X corresponde a la población de conejos y el eje Y a la de los zorros. La evolución del sistema comienza a partir de una condición inicial, es decir, un valor inicial para cada una de las poblaciones (digamos $X(t=0)=X_0$ e $Y(t=0)=Y_0$). Esta condición inicial aparece representada en el espacio de fases con un punto $(X_0,\, Y_0)$. La evolución temporal del sistema hace que las poblaciones cambien con el tiempo, de modo que tendremos unas funciones $X(t)$ e $Y(t)$. De este modo, en cada instante de tiempo el estado de sistema estará representado en el espacio de fases por un punto $(X(t),\, Y(t))$ y la evolución temporal dará lugar a una trayectoria $\mathbf{S}(t)=(X(t),\, Y(t))$.[1]

Cuando esta trayectoria se aproxima en el límite de tiempos muy grandes (esto es, asintóticamente) hacia una serie de estados finales, a este conjunto de estados finales se les denomina *atractor*. Podemos distinguir tres clases principales de comportamiento asintótico:

1. **Comportamiento estable**. Para cualquier condición inicial, el estado del sistema evoluciona con el tiempo bien hacia un *punto fijo* $(\bar{X},\, \bar{Y})$, esto es, hacia un valor constante de X y de Y (también denominado *punto límite* o *punto estable*), o bien hacia un *ciclo límite*, esto es, hacia un comportamiento periódico en el que los estados se repiten periódicamente al cabo de un cierto tiempo, denominado periodo. Se le denomina así por la forma circular que alcanza la trayectoria después de un tiempo de evolución. Los atractores de los sistemas dinámicos estables son puntos fijos o ciclos límite. Un mismo sistema puede tener más de un atractor. Esto quiere decir que, dependiendo de dónde comencemos, esto es, de cuál sea nuestro estado inicial, acabaremos en un atractor o en otro. Hablamos entonces de *cuencas de atracción* para referirnos al conjunto de estados iniciales con el mismo atractor.

2. **Comportamiento inestable**. La evolución del sistema escapa a los posibles atractores que pueda tener, por lo que el estado del sistema se aleja indefinidamente de la condición inicial. Un mismo sistema dinámico puede mostrar un comportamiento estable para unas condiciones iniciales, e inestable para otras. Por esta razón son muy importantes los análisis de estabilidad en los sistemas dinámicos, ya que nos permiten dividir el espacio de fases en dominios de estabilidad e inestabilidad con respecto a los posibles atractores del sistema.

3. **Comportamiento caótico**. Un sistema muestra un comportamiento caótico cuando, al cabo de un cierto tiempo, el estado del sistema permanece confinado en una zona del espacio de fases (mostrando de esta forma un comportamiento estable), pero sin tender a un atractor fijo, esto es, ni hacia un punto fijo ni hacia un ciclo límite. Este comportamiento podría ser considerado inestable, por lo que se puede dicir que un sistema se comporta caóticamente cuando exhibe los dos tipos de comportamiento de una manera compleja. En los sistemas caóticos, el sistema comienza a visitar los estados del atractor, denominado en este caso *atractor extraño*, saltando de un punto a otro de una forma completamente desordenada y sin ninguna periodicidad definida (se suele decir que el periodo es infinito). Estos atractores extraños suelen tener una estructura fractal, razón

[1] Formalmente, esta trayectoria es una curva paramétrica ya que los puntos dependen implícitamente del parámetro t.

por la cual se suele decir que los fractales son la representación o la imagen del caos.

La característica fundamental que diferencia al comportamiento caótico, o *caos*, del resto de sistemas dinámicos es la *impredecibilidad*. En el caso de los sistemas estables sabemos con probabilidad 1 hacia qué estado tiende el sistema independientemente de la condición inicial. Los sistemas inestables, sin embargo, muestran una gran dependencia con las condiciones iniciales. Una pequeña variación en las condiciones iniciales del proceso provoca que las trayectorias diverjan en el tiempo, es decir, que se vayan separando. Supongamos que en nuestro problema de los conejos y los zorros partimos de una condición inicial ligeramente distinta $(X_0+\delta X_0,\, Y_0+\delta Y_0)$, siendo δX_0 y δY_0 cantidades muy pequeñas, y lo dejamos evolucionar dando una nueva trayectoria $\mathbf{S}'(t)$. Si el sistema es inestable lo que observaremos es que la distancia entre las dos trayectorias $\delta S(t) = |\mathbf{S}'(t) - \mathbf{S}(t)|$, que inicialmente era muy pequeña $\delta S(0) = |(\delta X_0,\, \delta Y_0)|$ se amplifica en el tiempo de forma exponencial, más o menos como $\delta S(t) \simeq \delta S(0) \times \exp(\lambda t)$, donde λ es un exponente característico del sistema denominado *exponente de Lyapunov*. Exponentes negativos (o nulos) de Lyapunov producen comportamientos estables: las trayectorias convergen en el tiempo (o su distancia permanece acotada), mientras que exponentes positivos son indicadores de inestabilidad: trayectorias inicialmente muy próximas se separan con el tiempo. Sin embargo a pesar de la importante dependencia de los sistemas inestables con las condiciones iniciales, es posible conocer, muchas veces con exactitud, su evolución en el tiempo, es decir, podemos predecir en qué estado se encontrará el sistema en cualquier momento de su evolución.

En el caso de los sistemas caóticos la impredecibilidad es total: una mínima diferencia en las condiciones iniciales hace que el sistema evolucione de una manera totalmente distinta y absolutamente impredecible. Esto es debido a que los exponentes de Lyapunov varían de una forma extremadamente compleja en función del punto del espacio de fases en el que nos encontremos: dos puntos del espacio de fases muy próximos entre sí pueden tener exponentes de Lyapunov completamente diferentes. Estos sistemas no admiten una solución exacta, es decir, no existe una función analítica para la solución $(X(t),\, Y(t))$. Por consiguiente, es necesario recurrir a métodos numéricos en su solución. Como cualquier tipo de método numérico conlleva aproximaciones o redondeos, por muy pequeños que estos sean, esto es, por muy preciso que sea el método, y aunque conozcamos con exactitud las condiciones iniciales, pasado un cierto tiempo nuestra solución será completamente distinta a la verdadera. Esta impredecibilidad choca conceptualmente con la naturaleza determinista del sistema, es decir, a pesar de que el estado futuro del sistema está determinado por sus condiciones iniciales, nunca podremos conocerlo.

El ejemplo típico (y uno de los primeros sistemas caóticos conocidos) es el modelo matemático de tres variables propuesto por el matemático y meteorólogo Edward N. Lorenz en 1963 para predecir el clima. Lorenz derivó este modelo simple a partir de ecuaciones simplificadas de la evolución de los rollos convectivos de la atmósfera y su estudio dio origen al denominado "*efecto mariposa*", una forma metafórica que se ha utilizado desde entonces para referirse a la sensibilidad de los sistemas caóticos a las condiciones iniciales. Según los resultados de Lorenz (1972), una perturbación extremadamente insignificante en las condiciones iniciales, como por ejemplo la variación en la temperatura y presión local producida por el aleteo de una mariposa en Brasil, podría provocar que la previsión del tiempo en Texas cambiase de un tiempo apacible

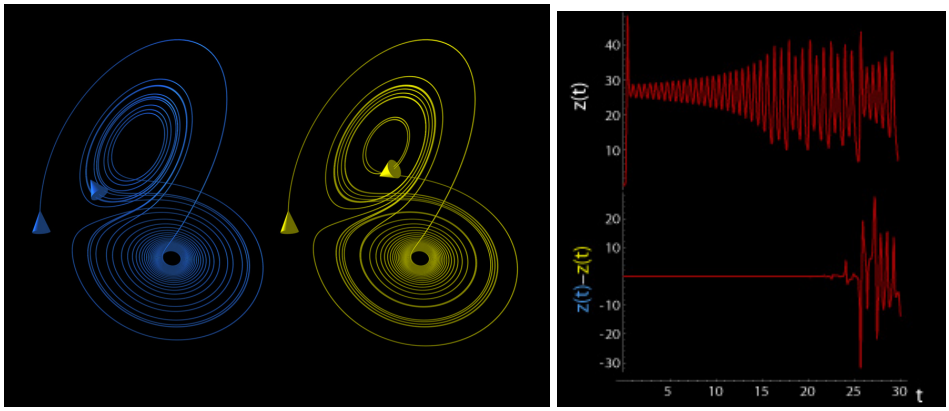


Figura 14.1: Atractor de Lorenz (izquierda) y efecto mariposa (derecha).

a un violento tornado. Caos es, por tanto, sinónimo de impredecibilidad, puesto que la predicción del fenómeno pasa por un conocimiento exacto y exhaustivo de las condiciones iniciales, lo cual es materialmente imposible en muchos casos prácticos. En el caso del clima, como sólo contamos con una cantidad finita de información sobre las condiciones iniciales, la predicción no pude extenderse más allá de una semana.

En la Fig. 14.1 se muestra el atractor extraño de Lorenz, resultado de la evolución temporal del modelo de Lorenz para el clima, y el efecto mariposa. En la figura de la izquierda se muestran dos trayectorias en el diagrama de fases tridimensional (puesto que el modelo cuenta con tres variables) que comienzan en dos puntos iniciales, indicados mediante los conos, que sólo difieren en la coordenada x en una cantidad de 10^{-5}. Como se puede apreciar, en ambos casos las trayectorias permanecen confinadas en la misma región del espacio de fases describiendo órbitas erráticas con diferentes amplitudes que parecen alternarse de modo caprichoso alrededor de dos puntos del espacio de fases. Esa estructura, que recuerda a las alas de una mariposa, es el conocido atractor extraño de Lorenz. En la figura de la derecha se muestra, arriba, la evolución temporal de la variable z para una de las trayectorias, y abajo, la variación temporal de la diferencia entre los valores de esta variable para las dos trayectorias. Inicialmente las dos trayectorias parecen coincidir (obsérvese la pequeña diferencia en la coordenada z). Sin embargo, para $t > 23$ la diferencia se hace tan grande como los valores que toma la propia variable durante la trayectoria, mostrados en la figura de arriba. La posición final para $t = 30$, indicada con los conos, muestra que las trayectorias ya no coinciden.

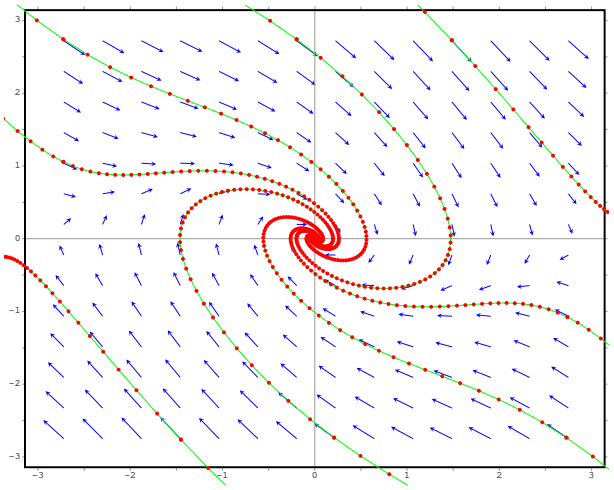
14.2. Sistemas dinámicos continuos y discretos

Un sistema dinámico es, básicamente, una regla de evolución. En física, un ejemplo de sistema dinámico es la ecuación del movimiento rectilíneo uniforme

$$x(t) = x_0 + vt$$

que describe cómo el estado $x(t)$, la posición de un móvil, varía en el tiempo t, debido a su movimiento con velocidad v. Otro ejemplo podría ser la ecuación diferencial equivalente

$$\frac{d}{dt}x(t) = v \tag{14.1}$$



La figura se ha generado con *Maxima* usando:

```
load("plotdf");
plotdf([y, -y-sin(x)],
       [nsteps, 100],  [tstep, 0.1],
       [xradius,3.1416], [yradius,3.1416]);
```

Figura 14.2: Diagrama de fases de la ecuación (14.3) de un péndulo amortiguado.

También la ecuación podría ser

$$x(n+1) = x(n) + v\Delta t \tag{14.2}$$

Las dos últimas expresiones serán las que encontremos más frecuentemente en física: usualmente no conoceremos más que la regla con la que nuestro sistema físico cambia de un estado al inmediatamente siguiente.

La diferencia entre las expresiones (14.2) y (14.1) es evidente: la primera describe un movimiento continuo; la segunda, un movimiento "a saltos" o discreto (o "estroboscópico"). El primer tipo de sistema dinámico diremos que describe un "flujo", mientras que del segundo diremos que es una aplicación o "mapa"[2]. Para entender el porqué de esta nomenclatura, veamos las figuras 14.2 y 14.3.

Flujos y mapas

En la figura (14.2) se muestra el flujo de la ecuación diferencial de segundo orden[3]

$$\ddot{x} + \beta\dot{x} + \sin x = 0 \tag{14.3}$$

Esta ecuación describe el movimiento de un péndulo sumergido en un fluido, una vez se han elegido convenientemente las unidades de longitud, tiempo y masa. En esta figura, los ejes de coordenadas son la posición x (en abscisas) y la velocidad $\dot{x}$ (en ordenadas): las dos variables que describen el estado del péndulo. En la figura aparece representado un campo de vectores "velocidad", cuya componente horizontal indica la

[2]El nombre de "*mapa*" se suele considerar un anglicismo. En Inglés, *map* significa aplicación, en el sentido matemático de transformación. Esto es, una correspondencia (con algunas restricciones) que lleva elementos de un conjunto a elementos de otro. Por eso se habla del "*mapa* logístico" en vez de la "aplicación logística". Aquí, sin embargo, emplearemos los dos términos de manera coloquial (y sin excesivo rigor matemático), ya que, a fin de cuentas un *mapa*, en Español, es también una *correspondencia* entre puntos geográficos y su representación en el papel.

[3]En los sistemas dinámicos casi siempre la variable independiente será el tiempo t. Por comodidad de notación representaremos las derivadas respecto al tiempo por puntos sobre las variables dinámicas. Así, la velocidad, $v = dx/dt$, será la primera derivada respecto al tiempo de la variable de posición x, y la denotaremos por $\dot{x}$. La aceleración será la segunda derivada, $a = d^2x/dt^2$, y la denotaremos por $\ddot{x}$.

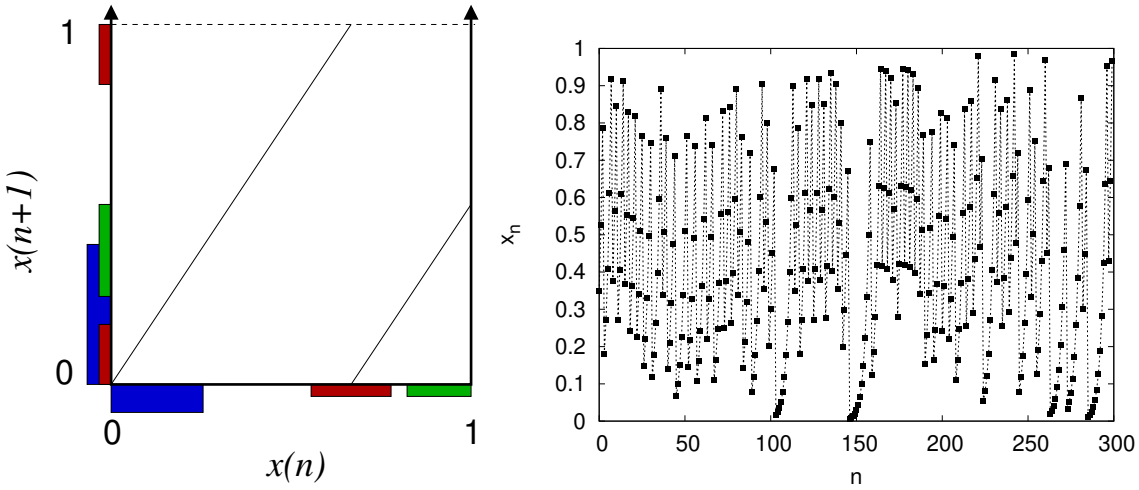


Figura 14.3: Transformación de intervalos (en el eje X) en intervalos (en el eje Y) y secuencia temporal de la aplicación dada por la ecuación (14.4).

velocidad de x (la velocidad de la partícula), mientras que su componente vertical indica la velocidad con que cambia $\dot{x}$ (la aceleración). Estos vectores son tangentes, en todo momento, a las posibles trayectoras en el espacio de fases, indicadas por puntos en la figura. Está claro que esta figura se parece a las líneas que forma la corriente en fluido (piénsese en la superficie del café cuando lo removemos), de ahí el nombre de "flujo" para este tipo de sistemas. Habrá puntos que permanezcan fijos (los "centros de los remolinos"), habrá líneas de corriente que se cierren (formando bucles o "ciclos límite"), etc.

En la figura 14.3 se muestra un ejemplo de la transformación de un intervalo por la aplicación

$$x(n+1) = \frac{3}{2}x(n) \bmod 1 \tag{14.4}$$

Ésta es una aplicación de un conjunto en un conjunto que transforma las posiciones de sus puntos, cambiando la forma de los intervalos en que se agrupan estos. La reiteración de esta aplicación puede, como en este ejemplo, "romper" el intervalo en "pedazos", mezclar estos "pedazos", dejar algún punto fijo (sin cambiar), etc. y generar una secuencia totalmente desordenada.

Antes de continuar, aclaremos que a todo flujo le corresponde una aplicación tal que su estudio es suficiente para describir el flujo. Esta aplicación se llama aplicación de Poincaré, en honor al padre del estudio de los sistemas dinámicos (Henri Poincaré, matemático francés, 1854-1912). Esta idea simplifica mucho el estudio matemático de los sistemas dinámicos y nos aprovecharemos de ello en este tema, en el que no trataremos los flujos.

Descripción cualitativa de un sistema dinámico

Como vimos en la introducción, un sistema dinámico (flujo o mapa) puede mostrar los siguientes comportamientos cualitativos:

1. Puede tener estados que no cambien al avanzar el tiempo.

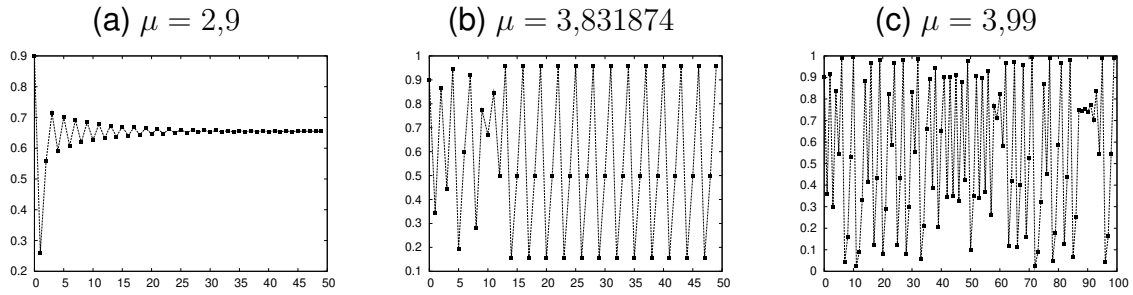


Figura 14.4: Evoluciones dadas por iteración la ecuación (14.5): punto fijo, período 3 y caos.

2. Puede tener estados que cambien al avanzar el tiempo, pero que vuelvan a su valor inicial tras un intervalo que llamaremos periodo.

3. Puede tener estados que se repitan de manera aproximada cada cierto intervalo, pero nunca exactamente.

4. Puede tener estados que no se repitan nunca, ni siquiera de manera aproximada.

Esta clasificación tan evidente de los estados de un sistema dinámico introduce los conceptos de punto fijo (1), órbita periódica (2), cuasiperiodicidad (3) y caos (4). Estos conceptos son intuitivos (tras un breve trato con los sistemas dinámicos) como también lo son los conceptos de estabilidad e inestabilidad asociados a ellos. Un punto fijo o una órbita se denominan estables cuando, al avanzar el tiempo, el sistema se acerca a ellos; en caso contrario, se dirá que son inestables.

El punto fijo de la figura 14.2 es, por ejemplo, estable: representa el final del movimiento del péndulo en el fluido, estado en el que éste se halla en reposo.

En la figura 14.4 se muestran tres evoluciones del sistema dinámico[4]

$$x_{n+1} = \mu x_n (1 - x_n) \tag{14.5}$$

Este sistema dinámico lo utilizaremos mucho en lo que sigue: se llama aplicación logística o mapa logístico. Se trata de uno de los primeros y más conocidos sistemas dinámicos discretos y fue propuesto para modelar la evolución temporal de una población. Es el ejemplo arquetipo de cómo un comportamiento complejo y caótico puede aparecer en un sistema muy simple.

En la figura 14.4(a) se muestra una evolución hacia un punto fijo que, por lo tanto, es estable. En la figura 14.4(b) se muestra una órbita de período 3, hacia la que tiende el sistema, por lo que es una órbita estable. En la figura 14.4(c) se muestra una "órbita" caótica. A pesar de no repetirse nunca, visita puntos en las cercanías de un conjunto que, se puede decir, es estable, ya que el sistema nunca se aleja de él: este conjunto es un "atractor".

El cambio de un comportamiento a otro se llama bifurcación y depende sólo del valor del parámetro μ. Existen valores críticos de μ en los que, por ejemplo, el punto fijo que se ve en 14.4(a) deja de ser visible. La razón es que ese punto deja de ser estable y el sistema se aleja de él. A cambio, en el caso del sistema (14.5), aparece una órbita de período 2. Esta órbita sigue siendo visible (es decir, es estable) según

[4] Por conveniencia, en el caso de sistemas discretos se utiliza habitualmente la notación de subíndices para los "pasos de tiempo".

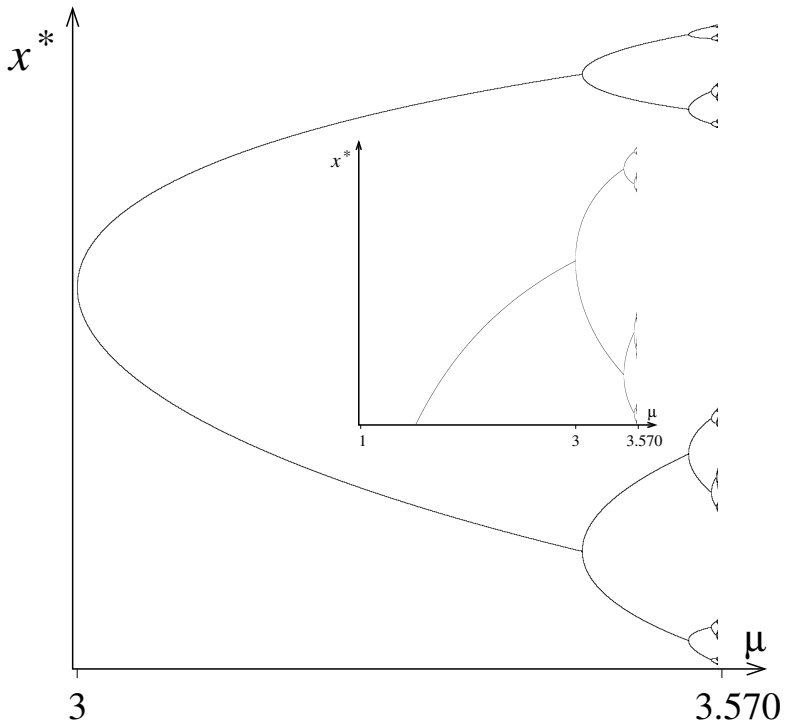


Figura 14.5: Diagrama de bifurcaciones de la aplicación logística (14.5) para valores del parámetro $\mu \in [1;\, 3{,}569945672\ldots)$.

aumentamos el valor del parámetro μ, hasta que pierde su estabilidad; en el caso de la ecuación logística, aparece una órbita de período 4 en su lugar. Se produce una duplicación de período. Este comportamiento se observa una y otra vez, cada vez ante cambios más pequeños del parámetro μ.

Si representamos sólo los puntos de las sucesivas órbitas estables de la aplicación, en función de los valores que demos al parámetro μ, observamos algo como la figura 14.5. Esta figura se denomina diagrama de bifurcaciones de la aplicación. El comportamiento de sucesivas duplicaciones de período es la cascada de bifurcaciones de Feigenbaum (en honor a este físico estadounidense, que la describió por primera vez en 1975, gracias a cálculos numéricos con un ordenador, como vamos a hacer nosotros).

¿Qué sucede a la derecha de este diagrama de bifurcaciones de la cascada de Feigenbaum? La respuesta es: "el caos"... y las ventanas periódicas. La continuación se muestra en la figura 14.6. Las zonas densamente pobladas de puntos se denominan bandas caóticas. Estas bandas comienzan a la derecha del punto donde acaba la cascada de duplicación de período de Feigenbaum. Este punto se llama *punto de acumulación de Myrberg-Feigenbaum*, y se representa por $\mu_\infty = 3{,}569945672$. Dentro de las bandas caóticas hay zonas de regularidad. Se puede encontrar, por ejemplo, la ventana de período 3. Dentro de esta ventana de período 3 se adivina la forma de la cascada de duplicación de Feigenbaum, que dota a este diagrama de bifurcaciones de estuctura fractal.

En la siguiente sección estudiaremos brevemente la aplicación logística y su diagrama de bifurcaciones mediante distintos programas informáticos.

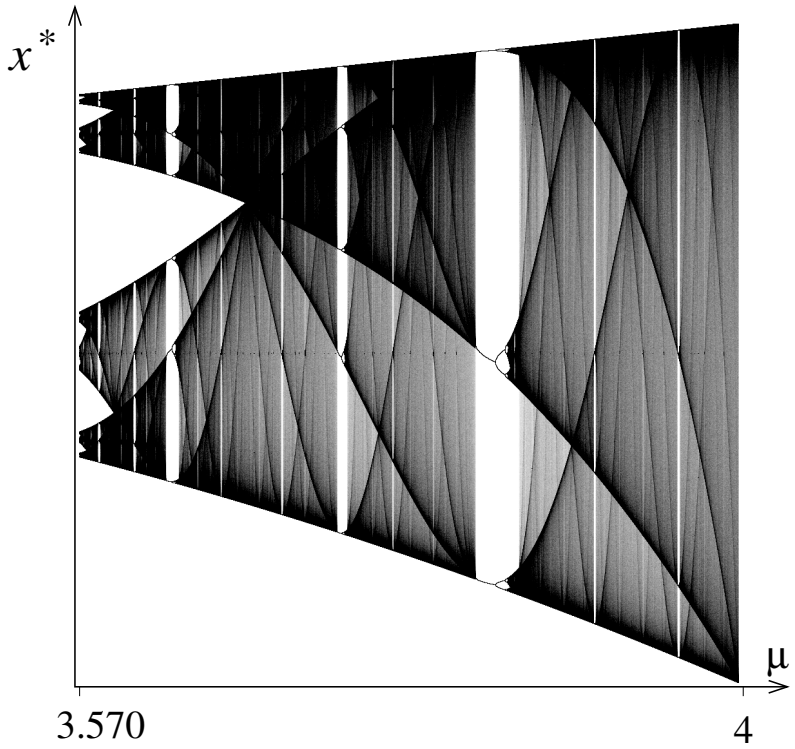


Figura 14.6: Diagrama de bifurcaciones de la aplicación logística (14.5) para valores del parámetro $\mu \in [3{,}569945672;\ 4)$.

14.3. Aplicaciones unidimensionales: Aplicación logística

La aplicación logística (14.5) se define en el programa como una función C con dos argumentos: uno es el parámetro μ (que llamaremos por su nombre "`mu`") y otro es el argumento de la aplicación, x. La función que iteraremos es: "`double f(double mu, double x)`".

Tanto el valor del parámetro ("`mu`") como el número de iteraciones que se quieren guardar ("`niter`") se indican en los argumentos del programa (luego se usan "atof" y "atoi" para interpretar esos argumentos).

Para que la iteración haya convergido hacia una órbita estable, en caso de que ésta exista para el valor de μ seleccionado del programa, se ejecuta la iteración de la aplicación logística "`NITER0`" veces: esto garantiza una buena convergencia (mejor cuanto más grande sea "`NITER0`"). Finalmente se ejecuta la iteración indicada por "`niter`" y se exportan los datos como pares ordenados $(x_n,\ x_{n+1} = f(\mu;\ x_n))$.

El listado de este programa es el 14.1.

Listado 14.1: El programa que exporta un número de iteraciones de la aplicación logística para un valor dado del parámetro.

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <stdlib.h>
4
5 #define NITER0       2048
```

```
6
7  /* Mapa logístico */
8  double f(double mu, double x)
9  {
10         return mu*x*(1-x);
11 }
12
13 int main(int argc, char** argv)
14 {
15         int   n, niter;
16         double mu, x, y;
17
18         if( argc<3 )
19         {
20                 printf("Uso del programa:\n %s <parametro mu>"
21                             " <no. iteraciones>\n",
22                        argv[0]);
23                 exit(0);
24         }
25
26         mu=atof(argv[1]);
27         niter=atoi(argv[2]);
28
29         x=0.5;
30         for(n=0; n<NITER0; n++)
31         {
32                 x=f(mu, x);
33         }
34
35         for(n=0; n<niter; n++)
36         {
37                 y=f(mu, x);
38
39                 printf("%1.10f\t%1.10g\n",
40                        x, y);
41                 x=y;
42         }
43
44         return 0;
45 }
```

Ejercicio 14.1. Represente con Gnuplot la evolución de la variable x_n en función de n y compruebe que:

- para el valor de $\mu = 3{,}831874$ la aplicación logística tiene una órbita de período 3
- para los valores de $\mu = 3{,}49856170$ y $3{,}96027013$ la aplicación logística tiene órbitas de período 4

- para los valores de $\mu = 3{,}73891491$, $3{,}90570647$ y $3{,}99026705$ la aplicación logística tiene órbitas de período 5.

  Nota: Posiblemente sea necesario aumentar el valor del número de iteraciones de partida ("`NITER0`") para mejorar la convergencia a algunas de estas órbitas.

Ejercicio 14.2. Sustitúyase la función que define la aplicación logística por la aplicación

$$g(k;\, y) = 1 - ky^2$$

y compruébese que cuando k se toma como

$$k = \frac{1}{4}\left[(\mu - 1)^2 - 1\right]$$

la aplicación $g(k;\, y)$ tiene órbitas del mismo período que las de la aplicación logística para el correspondiente μ. La existencia de esta relación entre los parámetros indica que las aplicaciones $g(k;\, y)$ y $f(\mu;\, x)$ son *conjugadas*, esto es, que conocidas las órbitas de una de ellas, se conocen las de la otra.

Nota: Seleccionar como valor inicial de la iteración de $g(k;\, y)$ el valor $y = 0$.

14.3.1. Cálculo del período de una aplicación recursiva

En los ejercicios anteriores conocíamos el período de la aplicación logística para generar así el conjunto de puntos de la correspondiente órbita dada. De este modo se comprobaba que la órbita era del período propuesto.

En este apartado se busca que sea el programa el que determine el período de la órbita para un valor del parámetro μ dado. Para ello, lo que se propone es que el programa guarde los valores de un número N de iterados $\{x_i\}_{i=1}^{N}$ de la aplicación $f(\mu;\, x)$ y que, cuando calcule un nuevo x_n, compruebe si se halla tan próximo a alguno de ellos que se pueda considerar que la aplicación ha vuelto al mismo punto. Para ello definimos un radio ε de modo que, consideramos que dos valores de x_n coinciden cuando su distancia es inferior a ε. Esta distancia se calculará con la función "`fabs`" (valor absoluto en coma flotante), declarada en `<math.h>`.

Nota. Posteriormente, se puede comprobar esta hipótesis verificando que todos los puntos sucesivos se encuentran a distancias menores que ε de los valores históricos guardados $\{x_i\}_{i=1}^{N}$.

Para guardar los $\{x_i\}_{i=1}^{N}$ y, al mismo tiempo, ir comprobando si los x_n se encuentran próximos a ellos y, si no, guardándolos en el array pueden tomar dos aproximaciones: o bien se guardan N puntos y, cada vez que llega uno nuevo se desplazan los anteriormente guardados (se hace un *shift* de la lista), o bien se usa una "lista circular" de elementos. Una lista circular se implementa en C como un *array* lineal normal a cuyos elementos apuntan dos índices: uno de ellos puede crecer ilimitadamente; el otro, cuando supera el tamaño N del *array*, vuelve al inicio de éste. De este modo, el *array* aparece como ilimitado en el primer índice (el que realmente apunta a los elementos es el segundo), aunque sólo estará guardando los últimos N valores.

El listado 14.2 muestra el código del programa que hace esta evaluación del período.

Por supuesto, cuando μ se encuentre en una "ventana caótica" del diagrama de bifurcaciones del mapa logístico, el programa dará estimaciones erróneas. Lo mismo sucederá en pequeños entornos alrededor de los puntos de bifurcación, donde se está cambiando de un período a otro (al doble): en esos puntos el tiempo de convergencia a la órbita periódica se hace muy grande.

Listado 14.2: El programa que calcula el período de una órbita estable de la aplicación logística, dado su parámetro.

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <stdlib.h>
4
5  #define EPS          1e-8
6  #define NPUNTOS      100
7  #define CALENTAMIENTO 1500
8
9  double logistica(double x, double mu)
10 {
11         double xp;
12
13         xp=mu*x*(1-x);
14
15         return xp;
16 }
17
18 int periodoAplicacion( double (*f)(double,double),
19                        double *x0, double mu)
20 {
21         double x;
22         double xi[NPUNTOS];
23         int    i, iciclo;
24         int    j, jciclo;
25         int    periodo;
26
27         x=*x0;
28
29         /* iteraciones iniciales para llenar el array xi[] */
30         for(i=0; i<NPUNTOS; i++)
31         {
32                 x=f(x,mu);
33                 xi[i]=x;
34         }
35
36         /* iterar hasta encontrar parecidos */
37         periodo=0;
38         i=0;
```

```
39          iciclo=NPUNTOS-1;
40          do {
41                  x=f(x,mu);
42
43                  /* buscar parecidos en los puntos anteriores */
44                  jciclo=iciclo;
45                  for(j=NPUNTOS; j>0; j--)
46                  {
47                          if( fabs(xi[jciclo]-x)<EPS )
48                          {
49                                  periodo=NPUNTOS-j+1;
50                                  break;
51                          }
52
53                          jciclo--;
54                          if(jciclo<0)
55                                  jciclo=NPUNTOS-1;
56                  }
57
58          iciclo++;
59          if(iciclo==NPUNTOS)
60                  iciclo=0;
61          xi[iciclo]=x;
62
63          } while(periodo==0);
64
65          *x0=x;
66
67          return periodo;
68  }
69
70  int main(int argc, char** argv)
71  {
72          double x, mu;
73          int    i, periodo;
74
75          if( argc<2 )
76          {
77                  printf("Uso del programa:\n %s <parametro mu>\n",
78                          argv[0]);
79                  exit(0);
80          }
81
82          mu=atof(argv[1]);
83          x=0.5;
84
85          for(i=0; i<CALENTAMIENTO; i++)
86          {
```

```
87                x=logistica(x,mu);
88        }
89
90        periodo=periodoAplicacion(logistica, &x, mu);
91
92        printf("#mu= %0.8f\n", mu);
93        printf("#periodo= %d\n", periodo);
94        for(i=0; i<=periodo; i++)
95        {
96                x=logistica(x,mu);
97                printf("%d\t%0.8f\n", i, x);
98        }
99
100       return 0;
101 }
```

Ejercicio 14.3. Compruébese usando este nuevo programa que

- para el valor de $\mu = 3{,}831874$ la aplicación logística tiene una órbita de período 3
- para los valores de $\mu = 3{,}49856170$ y $3{,}96027013$ la aplicación logística tiene órbitas de período 4
- para los valores de $\mu = 3{,}73891491$, $3{,}90570647$ y $3{,}99026705$ la aplicación logística tiene órbitas de período 5.

¿Para qué valor de parámetro μ aparece, en la cascada de Feigenbaum, una órbita de período 8?

Nota: El programa del listado 14.2 tiene limitada su búsqueda de períodos a `NPUNTOS`, que es el tamaño del array cíclico y vale, en este caso, 100. Debido a las características del diagrama de bifurcaciones de la aplicación logística, las órbitas de este tamaño son muy difíciles de encontrar, ya que las ventanas en las que se dan son muy estrechas. Por eso, este valor de 100 no es una limitación importante del programa. En cualquier caso, es más importante la limitación del valor de `EPS` a algo "tan grande" (a veces) como 10^{-8}.

Ejercicio 14.4. Rescríbase el programa anterior realizando las búsquedas sobre una lista lineal (no cíclica). Cada nuevo valor de x_n calculado se guardará en la última posición, desplazando los anteriores hacia el comienzo de la lista, y eliminando el elemento más antiguo, en la posición 0 del array.

14.3.2. Diagrama de bifurcaciones de la ecuación logística

A pesar de que parece sólo una representación cualitativa, el diagrama de la figura 14.6 caracteriza todos sistemas dinámicos unidimensionales de la misma naturaleza que la aplicación logística. Entre estos sistemas se encuentran algunos como el sistema de Lorenz, básicos en la historia de la teoría del caos. La aplicación de Poincaré

del sistema de Lorenz tiene un diagrama de bifurcaciones similar al de la aplicación logística, y esto demuestra que ambos sistemas pertenecen a una misma universalidad: todos los comportamientos de bifurcaciones que se dan en la aplicación logística (y todas las conjugadas a ella) aparecerán en el sistema de Lorenz y, además lo harán, en el mismo orden, unos tras otros, que muestra el diagrama de bifurcaciones de la aplicación logística.

Ejercicio 14.5. El ejercicio que se propone aquí es generar una imagen en escala de grises (como se mostró en la sección 11.11) semejante al de la figura 14.6. El programa deberá solicitar (o recibir por línea de comandos) los valores de inicio μ_{ini} ($= 3{,}5$) y de final μ_{fin} ($= 4$) de la ventana de representación del parámetro. Por su parte, se puede aprovechar que el diagrama de bifurcaciones se "abre hacia la derecha" para calcular con el propio programa la escala de la imagen, esto es, los valores x_{min} y x_{max} que abarcará la imagen en su eje vertical para una representación óptima. Basta calcularlo al principio del programa para el μ_{fin} ($> \mu_{ini}$).

La forma más fácil de calcular los valores de gris de la imagen es la siguiente.

- A cada columna le corresponde un valor de μ que se interpola linealmente de modo que a μ_{ini} le corresponde la columna 0 y a μ_{fin} la columna `ANCHURA-1`.
- Se itera la aplicación logística y a cada x_n se le hace corresponder un píxel de la columna, de modo que a x_{min} le corresponda el píxel en la fila `ALTURA-1` y a x_{max} el píxel en la fila 0.
- En cada píxel de la matriz de enteros que proporcionará la imagen se cuenta el número de iteraciones que han caído en dicho píxel, hasta que uno de los píxeles alcance un número de repeticiones prefijado `MAXCUENTAS` (se puede tomar alrededor de 100, pero cuanto más grande sea, más "nítida saldrá la imagen").
- Cuando se termina con una columna, se transforman las cuentas en ella en niveles de gris en el intervalo `0...255`, mediante la transformación

$$n_g = 255 - (255 * \texttt{IMAGEN}_{ij})/\texttt{MAXCUENTAS}$$

  De este modo, los píxeles muy visitados serán negros y los poco visitados serán blancos; las órbitas periódicas serán líneas negras y las zonas caóticas o de transición tendrán zonas negras rodeadas de otras con amplios rangos de grises.

Para guardar la imagen resultante, se utilizarán las funciones ya escritas, y recopiladas en forma de biblioteca, y declaradas en el archivo H `<libguardaimagen.h>` que se muestra en el listado 14.3.

Listado 14.3: Declaraciones de las funciones que guardan una imagen en un archivo con formato `PGM`.

```
1 #ifndef _LIBGUARDAIMAGEN_H_
2 # define _LIBGUARDAIMAGEN_H_
3
```

```
4 #include <stdio.h>
5
6 /* Guarda en el archivo de nombre dado una imagen
7  * PGM de dimensiones anchura X altura que contiene en
8  * sus píxeles valores enteros entre pixel_min y pixel_max
9  */
10 void guardaPGMi(char* nombre, int anchura, int altura,
11                 int *pixels, int pixel_min, int pixel_max);
12
13 /* Guarda en el archivo de nombre dado una imagen
14  * PGM de dimensiones anchura X altura que contiene en
15  * sus píxeles valores reales entre pixel_min y pixel_max
16  */
17 void guardaPGMd(char* nombre, int anchura, int altura,
18                 double *pixels, double pixel_min, double pixel_max);
19
20 #endif
```

14.4. Aplicaciones bidimensionales: Conjunto de Mandelbrot

Del mismo modo que en la aplicación logística el comportamiento de las iteraciones depende del valor de μ, para la aplicación de Mandelbrot compleja el comportamiento va a depender del valor de un parámetro, sólo que éste es ahora un número complejo. La aplicación de Mandelbrot compleja es

$$z_{n+1} = z_n^2 + c$$

donde c es el parámetro (igual que lo eran μ o k en las aplicaciones unidimensionales que estudiamos anteriormente).

Los comportamientos que estudiaremos son la estabilidad o inestabilidad de la órbita que sigue la iteración en el plano complejo: una órbita será estable si se mantiene siempre acotada; será inestable si se "escapa al infinito". Comprobaremos esta calidad de la órbita con el siguiente test

$$|z_n| > R$$

donde n es el número de iteración ("suficientemente avanzada") y R un radio de divergencia "suficientemente grande". Por "suficientemente", vamos a tomar $R = 20$ unidades y un número de iteraciones $n < N$, con $N = 500$. Por lo general, no será necesario llegar a $n = N$ para que una órbita se escape del disco de radio R.

Se observa que el conjunto de puntos en los que cambia el comportamiento, de ser acotado a diverger (puntos de bifurcación) forman un conjunto conexo que tiene una estructura muy complicada. Este conjunto se llama conjunto de Mandelbrot, en honor de Benoit Mandelbrot (matemático Francés, aunque de origen polaco, nacido en 1924) que fue el primero en estudiarlo. La estructura de este conjunto es, además, fractal.

Para representar el conjunto de Mandelbrot, usualmente lo que se hace es representar los puntos internos al contorno determinado por este conjunto; esto facilita el

cálculo y la visualización. Sin embargo, hay que recordar que el conjunto de Mandelbrot, en sentido estricto, es la *frontera* de esa representación.

Ejercicio 14.6. Lo que se pide en este ejercicio es escribir un programa que itere la aplicación compleja de Mandelbrot (partiendo de $z_0 = 0$) en el rectángulo Re $c \in [-2;\, 0{,}5]$, Im $c \in [-1;\, 1]$, recorrido con cierta finura, digamos con pasos de $0{,}01$ tanto en la parte real como imaginaria. En función de que no se cumpla la condición de divergencia o sí, se escribirán en un fichero las coordenadas de ese punto c o no, respectivamente. Luego, con Gnuplot, se representará el conjunto de puntos resultante.

Nota: Para manipular los números complejos, se recomienda tratarlos como una estructura o tipo de dato, por ejemplo

```
struct complex {
        double re;
        double im;
};
```

Luego se necesitarán funciones que realicen tareas con estos números complejos (ya que el C no sabe cómo operar más que con sus campos de tipo `double`. Por ejemplo, el módulo (para evaluar la condición de divergencia) se podría definir como

```
double modulo(struct complex x)
{
        return sqrt(x.re*x.re+x.im*x.im);
}
```

Ejercicio 14.7. Modificar el programa anterior para que guarde no sólo el interior del conjunto de Mandelbrot, sino todos los puntos c comprobados junto con el número de iteraciones entre 1 y N necesarias para escapar del disco de radio R (o $N+1$ si no ha podido escapar).

Represéntese luego este conjunto de datos (guardado en el archivo "`mandelbrot2.dat`") con Gnuplot usando el comando

```
splot [-2:0.5][-1:1][0:100] 'mandelbrot2.dat' using 1:2:3 with pm3d
```

El resultado debería ser similar al de la figura 14.7.