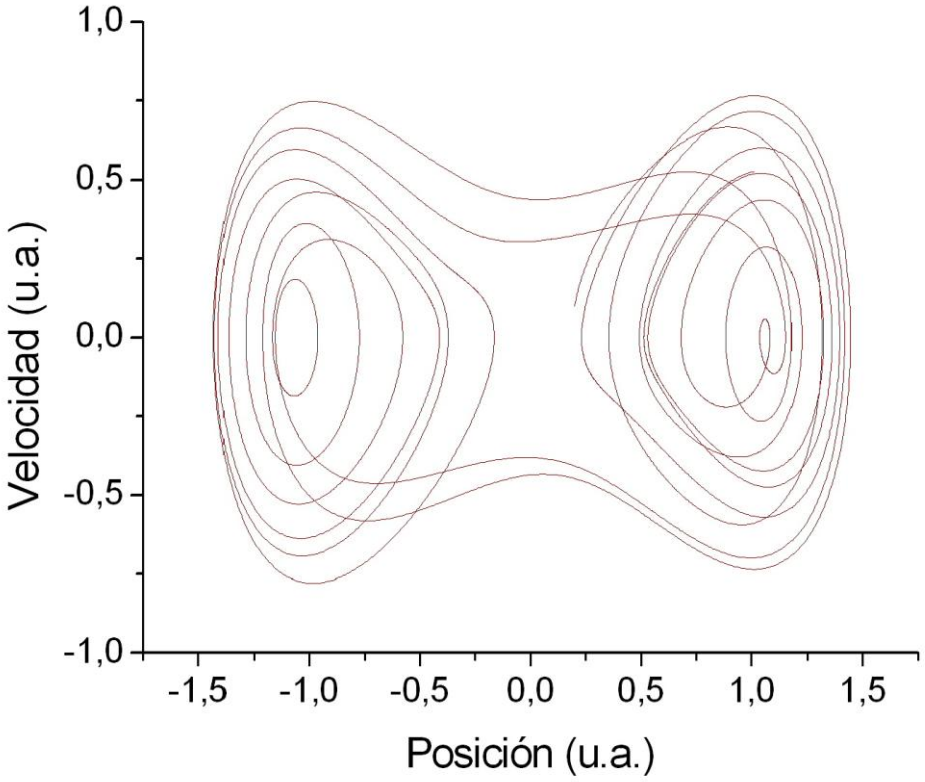
ESTUDIO DE LAS SOLUCIONES NUMÉRICAS DEL OSCILADOR DE DOBLE POZO



DAVID CASAS GARCÍA-MINGUILLÁN

AMPLIACIÓN DE MÉTODOS NUMÉRICOS

5º de Física

Estudio de las soluciones numéricas del oscilador de doble pozo

David Casas García-Minguillán
Edificio C-2, Campus de Rabanales, Universidad de Córdoba, 14071 España

9 de febrero de 2005

Resumen

En este artículo he calculado las soluciones numéricas, con distintas condiciones iniciales, de la ecuación de Duffing que rige el movimiento del oscilador de doble pozo. He utilizado el conocido método de Runge-Kutta de orden cuatro, programado en Fortran 90. Finalmente, comento de forma cualitativa el paso de oscilaciones regulares a caóticas en este oscilador no lineal.

1. Introducción

El oscilador de doble pozo es muy referenciado, como ejemplo de oscilador no lineal, en la literatura científica dedicada al estudio del caos, atractores extraños, soluciones aperiodicas, etc. Para determinadas soluciones se asemeja a un sistema cáotico, sensible a pequeñas perturbaciones externas y por tanto se comporta de forma impredecible, a pesar de estar definido por ecuaciones deterministas, como veremos a continuación[1].

En este artículo he hecho una simulación por ordenador del comportamiento del oscilador, sin embargo, aparte de los estudios computacionales, hay estudios experimentales hechos sobre este oscilador los cuales confirman el comportamiento caótico de este sistema[2].

A continuación expondré la descripción matemática del problema, es decir, el sistema de ecuaciones diferenciales con sus condiciones iniciales. Despues comentaré el método de Runge-Kutta, el cual lo he programado usando Fortran 90. Explicaré los programas y su funcionamiento. Calcularé tres casos particulares, y hablaré sobre los datos y gráficas obtenidos, los cuales dan un indicio del paso del orden al caos y rehaciendo los calculos para un rango de valores de una variable, mostraré donde cambia bruscamente de un comportamiento a otro. Este salto se visualiza mucho mejor en los videos que se encuentran en el CD-ROM que acompaña a este artículo. Son animaciones hechas a partir de muchas gráficas individuales[1], generadas con aproximadamente medio millón

[1] Realizadas a mano por el propio autor, lo que coloquialmente se denomina "trabajo de chinos".

de puntos distintos, en total. Finalmente, en el apéndice, he dejado escritos los códigos de los programas. En el CD-ROM, aparte de los videos, está grabado este mismo artículo, los dos programas en código fuente, los resultados de cada apartado en sus carpetas correspondientes y dos archivos que corresponden a las referencias 1 y 2 de este artículo.

2. Descripción matemática del problema

El oscilador de doble pozo puede describirse matemáticamente mediante la llamada *ecuación de Duffing*:

$$\frac{d^2x}{dt^2} + b\frac{dx}{dt} - x + x^3 = F\cos(\omega t) \tag{1}$$

en ella b es el coeficiente de amortiguamiento que afecta a la varilla en su oscilación, F es la amplitud de la fuerza periódica que afecta a todo el sistema, y ω es su frecuencia. Esta ecuación diferencial de orden dos, se puede convertir en un sistema de tres ecuaciones de orden uno[3]:

$$\begin{cases} x' = v \\ v' = -bv + x - x^3 + F\cos(z) \\ z' = \omega \end{cases} \tag{2}$$

Sin embargo, en este artículo, ω la vamos a considerar como una constante más, en concreto igual a 1. Por lo que el sistema se reduce a dos ecuaciones[2]:

$$\begin{cases} x' = v \\ v' = -bv + x - x^3 + F\cos(t) \end{cases} \tag{3}$$

Con dos condiciones iniciales: $x\,(t_0)$ y $v\,(t_0)$. El sistema va a tener dos puntos estables, $x = -1$ y $x = 1$, y un punto inestable, $x = 0$, cuya posición en el sistema físico se puede ver en la figura 1.

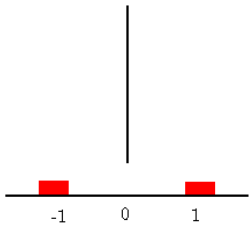


Figura 1: Situación de imanes y varilla.

[2] Dado que la frecuencia es el inverso del tiempo, la t que aparece dentro del coseno es adimensional: $t(u.a.) = 1(T^{-1}) * t(T)$ donde u.a. son unidades arbitrarias y T es la dimensión tiempo.

3. Métodos de Runge-Kutta

3.1. Introducción

El método de Runge-Kutta es uno de los existentes dentro de la clase de métodos de un solo paso, los cuales se pueden describir genericamente mediante las ecuaciones:

$$\begin{cases} y_0 = y\left(x_0\right) \\ y_{i+1} = y_i + h\phi(h, x_i, y_i),\ i = 0, 1, 2, ..., N-1 \end{cases} \tag{4}$$

donde ϕ es una función que caracteriza el método.

Para estimar la rapidez de convergencia de un método u otro, se dice que (4) es de orden p si para todo $x_i \quad a \leq x_i \leq b$, y para todo h suficientemente pequeño, existen constantes C y p tales que si

$$\tau_i = \frac{y\left(x_{i+1}\right) - y\left(x_i\right)}{h} - \phi\left(h, x_i, y\left(x_i\right)\right)$$

entonces

$$|\tau_i| = Ch^p \tag{5}$$

La constante C depende, en general, de la solución $y(x)$, de sus derivadas y de la longitud del intervalo sobre el que se busca la solución, pero es independiente de h[4].

Dentro de estos métodos están los de la serie de Taylor de la solución $y(x)$. Si $y^{p+1)}(x)$ es continua en $[a, b]$ entonces el desarrollo viene dado por:

$$\begin{cases} y\left(x_{i+1}\right) = y\left(x_i + h\right) \\ y\left(x_{i+1}\right) = y(x_i) + hy'(x_i) + \frac{h^2}{2!}y''(x_i) + ... + \frac{h^p}{p!}y^{p)}(x_i) + \frac{h^{p+1)}}{(p+1)!}y^{p+1)}(\xi_i) \end{cases} \tag{6}$$

Utilizando que $y'(x) = f(x, y(x))$, (6) puede escribirse en la forma:

$$y\left(x_{i+1}\right) = y(x_i) + h\left[f(x_i, y(x_i)) + ... + \frac{h^{p-1}}{p!}f^{p-1)}(x_i, y(x_i))\right] + hO(h^p) \tag{7}$$

Obtenemos un método de orden p considerando:

$$\phi\left(h, x_i, y\left(x_i\right)\right) = f(x_i, y(x_i)) + ... + f^{p-1)}(x_i, y(x_i))\frac{h^{p-1}}{p!} \tag{8}$$

3.2. Runge-Kutta

La idea general de este método es sustituir las derivadas de f presente en (8), por combinaciones lineales de la propia función f (en diferentes puntos) para obtener el orden requerido.

Por ejemplo, un algoritmo de Runge-Kutta de orden 2 es el llamado *método de Euler mejorado o de Heun*:

$$y_{i+1} = y_i + \frac{h}{2}\left[f(x_i, y_i) + f(x_i + h, y_i + hf(x_i, y_i))\right] \tag{9}$$

La mayor limitación de las fórmulas de Runge-Kutta es la cantidad de trabajo que requieren; este trabajo se mide en términos del número de veces que la función f se evalua. Para fórmulas de alto orden, el trabajo crece excesivamente; para los métodos de orden p con $p = 1, 2, 3$ y 4 el número de evaluaciones de f es justamente p, pero para $p = 5$, es necesario hacer 6 evaluaciones, para $p = 6$ hay que hacer 7, para $p = 7$ ya son 9 las que se necesitan, para $p = 8$ son 11, etc.

Como consecuencia, los métodos de orden menor a cinco con tamaño de paso pequeño son preferibles a los métodos de orden superior usando un tamaño de paso más grande.

Los métodos más utilizados son los de orden $p = 4$ entre ellos la *fórmula clásica de Runge-Kutta* de orden 4, esta se escribe en la forma

$$\begin{cases} y_{i+1} = y_i + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \\ k_1 = f(x_i, y_i) \\ k_2 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1) \\ k_3 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2) \\ k_4 = f(x_i + h, y_i + hk_3) \end{cases} \tag{10}$$

Este método es muy utilizado por su rapidez de convergencia y por su sencillez de fórmula. De hecho, este método es capaz de conseguir precisiones altas sin tener que tomar h tan pequeño como para hacer excesiva la tarea computacional o que los cálculos de redondeo planteen serias dificultades[5].

Cuando tenemos ecuaciones de orden superior a 1, se pueden reescribir en forma de sistemas de ecuaciones de primer orden, si tenemos derivadas de y hasta el orden m, el sistema sería:

$$\begin{cases} \frac{dy_1}{dx} = f_1\left(x, y_1, y_2, ..., y_m\right) \\ \frac{dy_2}{dx} = f_2\left(x, y_1, y_2, ..., y_m\right) \\ \qquad\qquad ..... \\ \frac{dy_m}{dx} = f_m\left(x, y_1, y_2, ..., y_m\right) \end{cases} \tag{11}$$

Escrito en notación vectorial, el algoritmo de Runge-Kutta definido en (10), con $\mathbf{Y} = (y_1, ..., y_m)$, $\mathbf{F} = (f_1, ..., f_m)$, $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_m)$, se escribiría[3]:

$$\begin{cases} \mathbf{Y}_0 = \boldsymbol{\alpha} \\ \mathbf{Y}_{i+1} = \mathbf{Y}_i + \frac{h}{6}(\mathbf{K}_1 + 2\mathbf{K}_2 + 2\mathbf{K}_3 + \mathbf{K}_4), \quad i = 0, 1, ... \\ \mathbf{K}_1 = \mathbf{F}(x_i, \mathbf{Y}_i) \\ \mathbf{K}_2 = \mathbf{F}(x_i + \frac{h}{2}, \mathbf{Y}_i + \frac{h}{2}\mathbf{K}_1) \\ \mathbf{K}_3 = \mathbf{F}(x_i + \frac{h}{2}, \mathbf{Y}_i + \frac{h}{2}\mathbf{K}_2) \\ \mathbf{K}_4 = \mathbf{F}(x_i + h, \mathbf{Y}_i + h\mathbf{K}_3) \end{cases} \tag{12}$$

Este será el método, que programaré para resolver numéricamente la ecuación del oscilador de doble pozo.

4. Resultados

4.1. Programa *RK4Oscilador*

Este sencillo programa genera dos ficheros, *posición.txt* y *velocidad.txt* con los resultados de los ídem. Los datos que hay que introducir son el tiempo de inicio, el de finalización, el numero de divisiones del intervalo (con estos tres datos hemos establecido la longitud del paso, h), las condiciones iniciales de posición y velocidad, el coeficiente de amortiguamiento y la amplitud de la fuerza. El comando TIMEF() sirve para evaluar el tiempo que tarda en ejecutarse la parte del programa situada entre las dos llamadas de esta orden. Además, al acabar, da en pantalla el ultimo valor del intervalo de la variable independiente[3], la posición, la velocidad, el valor del paso, h y el tiempo que ha tardado la parte ejecutora del programa. A continuación expondré los tres casos que he estudiado con este programa, explicando que valor he tomado para todas las variables, comentando las graficas del espacio de fases, de la posición y de la velocidad.

4.1.1. Primer caso

Los valores de las variables son los de la siguiente tabla:

Intervalo	$[0, 100]$
Divisiones	10000 ($h = 0{,}01$)
Posicion inicial	1
Velocidad inicial	0
coef. de amortiguamiento	0,25
amplitud de la fuerza	0,22

He elegido este primer caso para comprobar que no había cometido ningún error de bulto[4] al escribir el algoritmo. Para ello elegí un software comercial que resolviese una EDO de forma numérica y contrasté los resultados con los de mi programa. Usé el Scientific WorkPlace 5.0, el mismo que me ha servido para redactar este artículo, y esta es la tabla que compara uno y otro:

[3] Es la variable temporal del sistema físico. No hay que confundirla con el tiempo que tarda en ejecutarse el programa, tal y como se ha comentado antes.

[4] En una prueba preliminar con otra función puse en el algoritmo $h/2$ donde tendría que haber puesto h. El resultado no fue catastrofico, pero convergía a la solución verdadera mucho más lentamente.

Tiempo	PosicionSWP	PosicionFortran($h = 0{,}01$)
10	0,607 92	0,60792
20	1. 069 4	1,06941
30	0,878 77	0,87877
40	0,840 46	0,84046
50	1. 178 2	1,17822

desconozco que método numérico usa el SWP, hay que comentar, como se verá a continuación, que con estas condiciones el sistema oscila de forma estable alrededor del punto de equilibrio 1, de forma que las pequeñas diferencias de método y precisión no afectan significativamente al resultado. Posiblemente, en una región caótica no deban coincidir, aunque no lo he comprobado.

Las gráficas y la pantalla de ejecución son las siguientes:

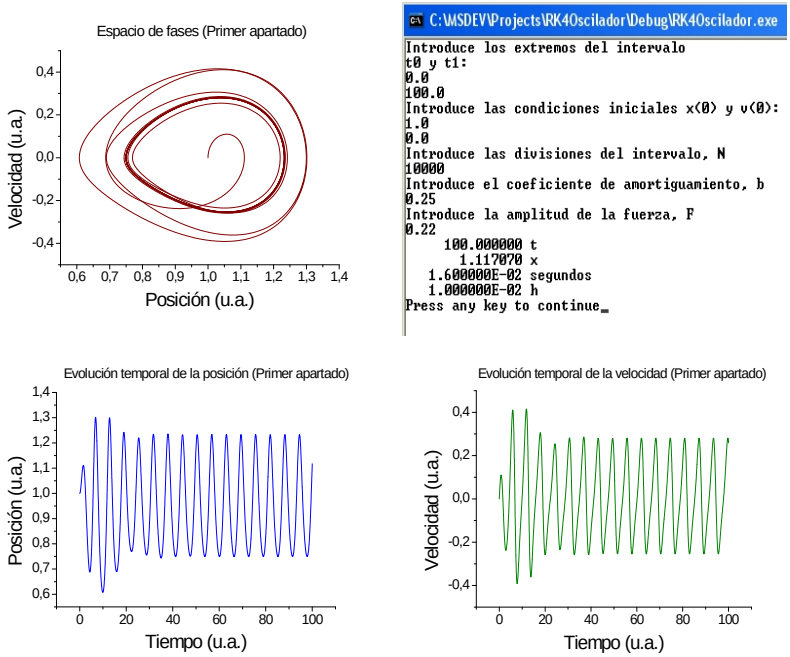


Figura 2

A partir de estas gráficas se puede concluir que con estos valores, el sistema, conforme avanza el tiempo, acaba describiendo un movimiento periódico en el espacio de fases con una órbita centrada en el punto de equilibrio 1.

4.1.2. Segundo caso

Los valores de las variables son los de la siguiente tabla:

Intervalo	$[0, 100]$
Divisiones	10000 ($h = 0{,}01$)
Posicion inicial	0,2
Velocidad inicial	0,1
coef. de amortiguamiento	0,25
amplitud de la fuerza	0,25

Las gráficas y la pantalla de ejecución son las siguientes:

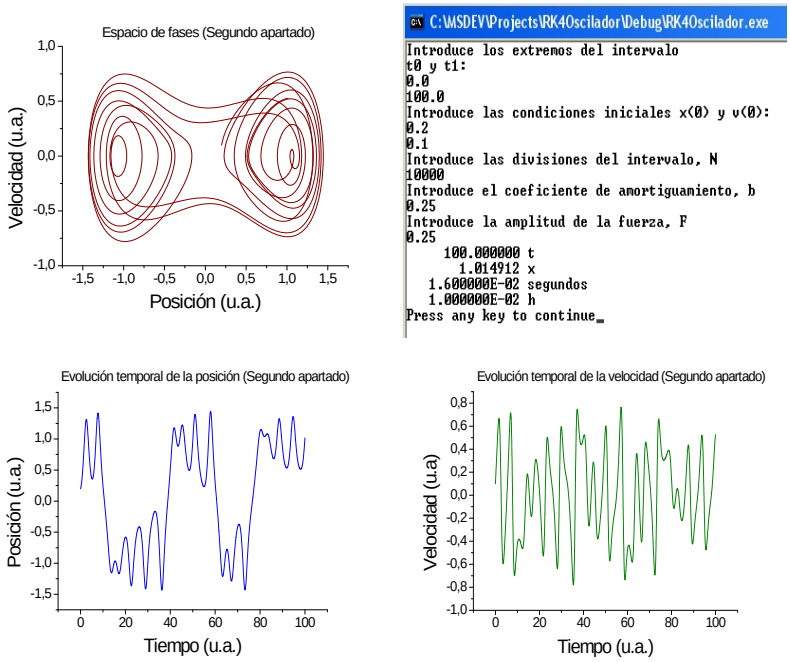


Figura 3

Se ve en el espacio de fases que el sistema ya ha empezado a entrar en un comportamiento caótico, las órbitas, aunque oscilantes en torno a los dos puntos de equilibrio, -1 y 1, ya no son predecibles. Como se verá, cuando comente el otro programa utilizado, el punto en el que el sistema salta de la predictibilidad al caos, no depende sólamente de las variables propias del sistema (fuerza, coeficiente, posición y velocidad inicial) que elijamos, sino tambien de la forma en la que ejecutemos el programa (el paso h, el número de iteraciones, etc.)

4.1.3. Tercer caso

Los valores de las variables son los de la siguiente tabla:

Intervalo	$[0, 100]$
Divisiones	10000 ($h = 0{,}01$)
Posicion inicial	0,2
Velocidad inicial	0,1
coef. de amortiguamiento	0,25
amplitud de la fuerza	0,4

Las gráficas y la pantalla de ejecución son las siguientes:

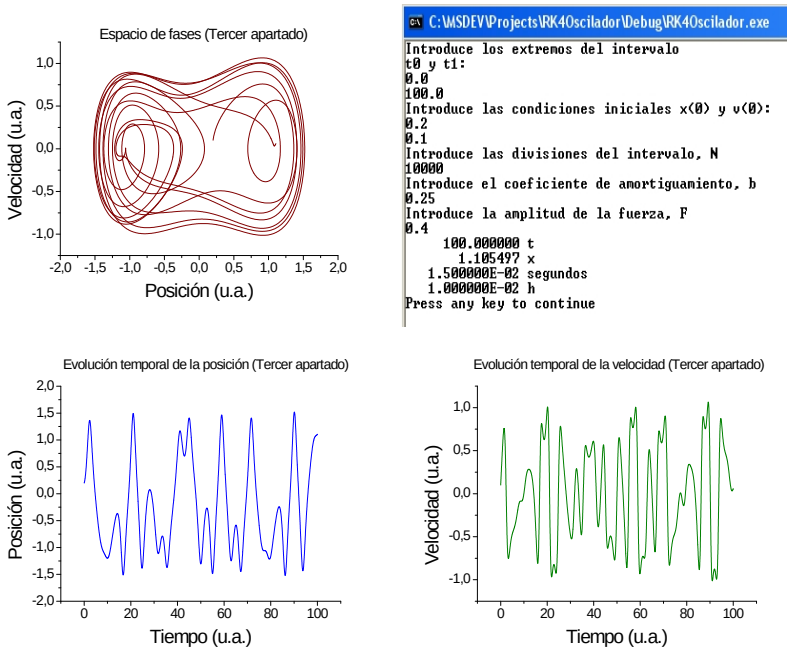


Figura 4

Se ve en el espacio de fases que el sistema ya entró en comportamiento caótico puro con órbitas que convergen en unos tramos y divergen en otros de forma completamente impredecible. El salto de soluciones períodicas a no periódicas se realiza de forma brusca y es dificil precisar donde ocurre. Por eso modifiqué el programa principal, con el que he calculado estos tres casos, para estudiar un poco más en detalle como es el paso del orden al caos.

4.2. Programa *RK4ODP*

Quería visualizar como ocurría el cambio del orden al caos con los datos de los casos dos y tres, variando únicamente la fuerza. Por eso modifiqué el programa, aunque el algoritmo de Runge-Kutta es el mismo, lo único es que se evalua

más veces. Introduzo un intervalo de variación de la fuerza, y el programa hace los mismos cálculos que el anterior para cada valor de fuerza. Luego graba los resultados en matrices de tres dimensiones (el anterior los grababa en matrices de dos) y escribe los resultados en los siguientes ficheros: *fuerza.txt*, *posición.txt*, *velocidad.txt* y *fases.txt*. Tanto este programa como el anterior graba todos los números en un único vector columna, ya que así se pueden grabar grandes cantidades de números sin problema. Luego el programa con el que construyo las gráficas, el Microcal Origin 6.0, al importar el fichero ASCII tiene la opción de interpretar los caracteres no numericos como inicio de una nueva columna, de forma que los ordena en formato de hoja de datos.

Ejecuté el programa con un valor de inicio de la fuerza de 0,20 y un valor final de 0,45 en 50 intervalos, con lo cual variaba en pasos de 0,005. En las siguientes cuatro gráficas vemos como salta del orden al caos, en las posiciones, entre $F = 0{,}26$ y $F = 0{,}265$:

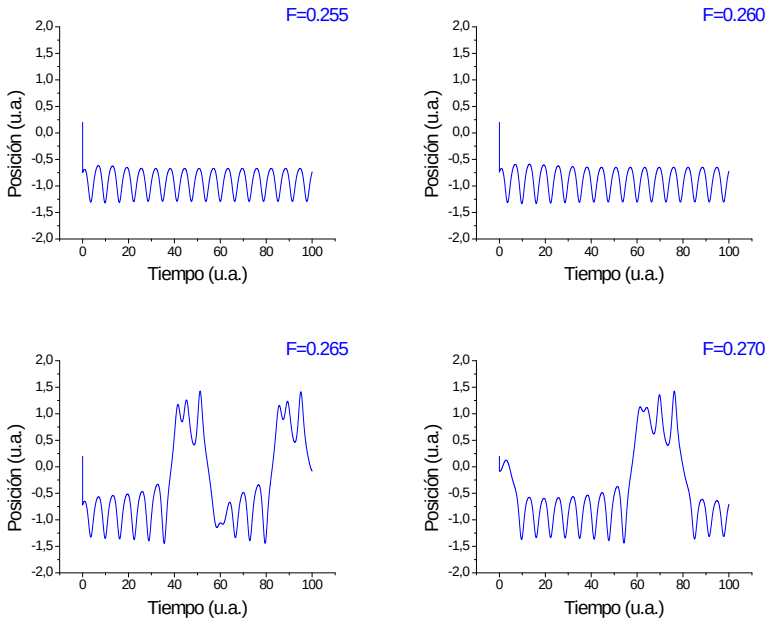


Figura 5

En el espacio de fases, con $F = 0{,}26$ y $F = 0{,}265$:

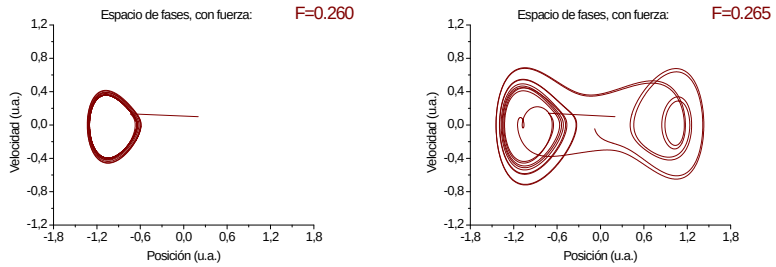


Figura 6

Vemos la pantalla de ejecución:

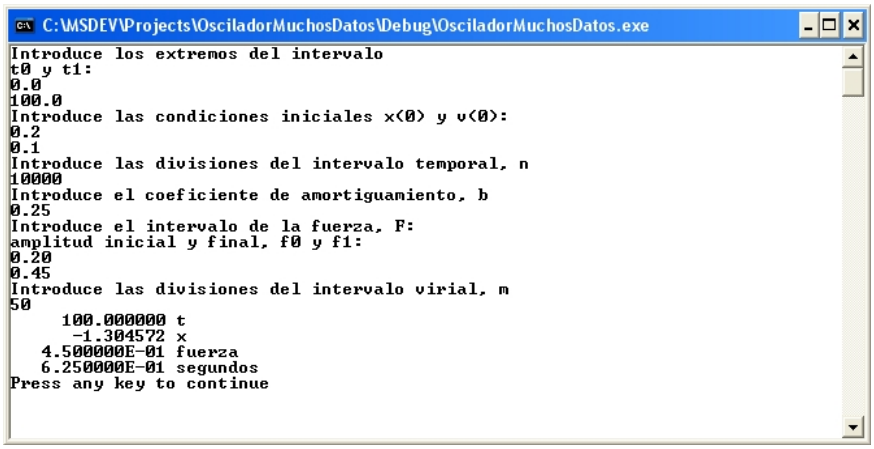


Figura 7

El tiempo de ejecución es de 0.625 segundos, mientras que en los tres casos anteriores era de 0.016, lo cual es debido a que tiene que repetir el mismo algoritmo para todos los valores de la fuerza. Con las 51 gráficas generadas para posición y 51 para espacio de fases, he construido dos videos, donde la sensación de movimiento debida al cambio en cada gráfica individual, ilustra mejor el comportamiento del oscilador.

5. Conclusiones

He obtenido las soluciones numéricas de la ecuación de Duffing aplicadad al oscilador de doble pozo usando el método de Runge-Kutta de orden 4. Aplicandolo a distintos casos, con diferentes valores de las variables y condiciones

iniciales, he mostrado como las soluciones cambian de la periodicidad a la aperiodicidad e impredictibilidad. Cuando el sistema se encuentra en el régimen caótico, es muy sensible a las condiciones iniciales, de forma que hacía falta dar pasos muy finos para precisar donde ocurria. Eso es lo que he hecho, sin embargo hay que matizar que lo he hecho para una sección muy concreta del espacio de fases *total*, donde están representadas todas las variables y condiciones iniciales del problema, como son t, x, v, $x(t_0)$, $v(t_0)$, b, F, h, t_0, t_1e incluso ω que aunque la hemos obviado, también interviene en la ecuación original. Intuitivamente se puede adivinar el comportamiento del sistema, ya que si sabemos que parte de un punto estable, con poca velocidad inicial, con un coeficiente amortiguador distinto de cero y una fuerza no muy grande, oscilará en torno al punto de equilibrio. Podemos por tanto, intentar adivinar que zonas pueden ser de interés para estudiarlas, ya que la simulación total, variando todo lo que puede variar, puede generar una cantidad astronómica de números, que para ser tratados necesitarían un gran tiempo de computación incluso en un superordenador.

Sin necesidad de ser tan ambicioso este artículo muestra de forma didáctica y aproximada el paso del orden al caos en un oscilador no lineal.

6. Apéndice

6.1. Código del programa *RK4Oscilador*

```
PROGRAM RK4Oscilador
USE PORTLIB
!Este es un programa para probar el metodo
!de Runge-Kutta de orden 4 en el caso del
!Oscilador de Doble Pozo
IMPLICIT NONE
REAL::b,h,f,t,x,v,tiempo,t0,t1
INTEGER::i,j,n
REAL,ALLOCATABLE,DIMENSION(:,:)::posres,velres
REAL,DIMENSION(4)::k,l
PRINT *, "Introduce los extremos del intervalo"
PRINT *, "t0 y t1:"
READ *,t0
READ *,t1
PRINT *, "Introduce las condiciones iniciales x(0) y v(0):"
READ *,x
READ *,v
PRINT *, "Introduce las divisiones del intervalo, N"
READ *,n
PRINT *, "Introduce el coeficiente de amortiguamiento, b"
READ *,b
PRINT *, "Introduce la amplitud de la fuerza, F"
```

```
READ *,f
ALLOCATE(posres(n+1,2),velres(n+1,2))
h=(t1-t0)/n
t=t0
posres(1,1)=t
posres(1,2)=x
velres(1,1)=t
velres(1,2)=v
tiempo=TIMEF()
DO i=1,n
k(1)=v
l(1)=-b*v-x*x*x+x+f*cos(t)
DO j=1,2
k(j+1)=v+l(j)*h/2
l(j+1)=-b*(v+l(j)*h/2)-(x+k(j)*h/2)*(x+k(j)*h/2)*(x+k(j)*h/2)+(x+k(j)*h/2)+f*cos(t+h/2)
END DO
k(4)=v+l(3)*h
l(4)=-b*(v+l(3)*h)-(x+k(3)*h)*(x+k(3)*h)*(x+k(3)*h)+(x+k(3)*h)+f*cos(t+h)
x=x+(h/6)*(k(1)+2*k(2)+2*k(3)+k(4))
v=v+(h/6)*(l(1)+2*l(2)+2*l(3)+l(4))
t=t0+h*i
posres(i+1,1)=t
posres(i+1,2)=x
velres(i+1,1)=t
velres(i+1,2)=v
END DO
tiempo=TIMEF()
PRINT *,t,"t"
PRINT *,x,"x"
PRINT *,tiempo,"segundos"
PRINT *,h,"h"
OPEN(8,file="posicion.txt")
DO j=1,2
     DO i=1,n+1
     WRITE (8,*) posres(i,j)
     END DO
     WRITE (8,*) 'a'
END DO
CLOSE(8)
OPEN(9,file="velocidad.txt")
DO j=1,2
     DO i=1,n+1
     WRITE (9,*) velres(i,j)
     END DO
     WRITE (9,*) 'a'
END DO
```

```
CLOSE(9)
END PROGRAM RK4Oscilador
```

6.2. Código del programa *RK4ODP*

```
PROGRAM RK4ODP
USE PORTLIB
!Este es un programa para probar el metodo
!de Runge-Kutta de orden 4 en el caso del
!Oscilador de Doble Pozo
IMPLICIT NONE
REAL::b,h,f,f0,f1,p,t,x,v,tiempo,t0,t1
INTEGER::i,j,g,n,m
REAL,ALLOCATABLE,DIMENSION(:,:,:)::posres,velres,fases
REAL,ALLOCATABLE,DIMENSION(:)::fuerza
REAL,DIMENSION(4)::k,l
PRINT *, "Introduce los extremos del intervalo"
PRINT *, "t0 y t1:"
READ *,t0
READ *,t1
PRINT *, "Introduce las condiciones iniciales x(0) y v(0):"
READ *,x
READ *,v
PRINT *, "Introduce las divisiones del intervalo temporal, n"
READ *,n
PRINT *, "Introduce el coeficiente de amortiguamiento, b"
READ *,b
PRINT *, "Introduce el intervalo de la fuerza, F:"
PRINT *, .ªmplitud inicial y final, f0 y f1:"
READ *,f0
READ *,f1
PRINT *, "Introduce las divisiones del intervalo virial, m"
READ *,m
ALLOCATE(posres(n+1,2,m+1),velres(n+1,2,m+1),fases(n+1,2,m+1),fuerza(m+1))
h=(t1-t0)/n
t=t0
p=(f1-f0)/m
DO g=1,m+1
posres(1,1,g)=t
posres(1,2,g)=x
velres(1,1,g)=t
velres(1,2,g)=v
fases(1,1,g)=x
fases(1,2,g)=v
END DO
tiempo=TIMEF()
```

```
DO g=1,m+1
f=f0+p*(g-1)
DO i=1,n
k(1)=v
l(1)=-b*v-x*x*x+x+f*cos(t)
DO j=1,2
k(j+1)=v+l(j)*h/2
l(j+1)=-b*(v+l(j)*h/2)-(x+k(j)*h/2)*(x+k(j)*h/2)*(x+k(j)*h/2)+(x+k(j)*h/2)+f*cos(t+h/2)
END DO
k(4)=v+l(3)*h
l(4)=-b*(v+l(3)*h)-(x+k(3)*h)*(x+k(3)*h)*(x+k(3)*h)+(x+k(3)*h)+f*cos(t+h)
x=x+(h/6)*(k(1)+2*k(2)+2*k(3)+k(4))
v=v+(h/6)*(l(1)+2*l(2)+2*l(3)+l(4))
t=t0+h*i
posres(i+1,1,g)=t
posres(i+1,2,g)=x
velres(i+1,1,g)=t
velres(i+1,2,g)=v
fases(i+1,1,g)=x
fases(i+1,2,g)=v
END DO
fuerza(g)=f
END DO
tiempo=TIMEF()
PRINT *,t,"t"
PRINT *,x,"x"
PRINT *,f,"fuerza"
PRINT *,tiempo,"segundos"
OPEN(8,file="posicion.txt")
DO g=1,m+1
DO j=1,2
      DO i=1,n+1
      WRITE (8,*) posres(i,j,g)
      END DO
      WRITE (8,*) 'a'
END DO
WRITE (8,*) 'pppp'
END DO
CLOSE(8)
OPEN(9,file="velocidad.txt")
DO g=1,m+1
DO j=1,2
      DO i=1,n+1
      WRITE (9,*) velres(i,j,g)
      END DO
      WRITE (9,*) 'a'
```

```
WRITE (9,*) 'vvvv'
END DO
END DO
CLOSE(9)
OPEN(10,file="fases.txt")
DO g=1,m+1
DO j=1,2
    DO i=1,n+1
    WRITE (10,*) fases(i,j,g)
    END DO
    WRITE (10,*) 'a'
WRITE (10,*) 'ffff'
END DO
END DO
CLOSE(10)
OPEN(11,file="fuerza.txt")
DO g=1,m+1
    WRITE (11,*) fuerza(g)
END DO
CLOSE(11)
END PROGRAM RK4ODP
```

Agradecimientos

Agradezco a Mercedes Marín Beltrán su ayuda y consejos en la elaboración de este trabajo.

Referencias

[1] **V Conferencia Nacional de Ciencias de la Computación (noviembre 98, Potosí)**, *Sistemas no lineales. Conceptos, algoritmos y aplicaciones*, José Manuel Gutiérrez

[2] **Chaos, Solitons & Fractals, Vol. 8, No 4, pp. 681-697**, *Routes to Chaos in a Magneto-Elastic Beam*, R.C.A. Thompson, T. Mullin

[3] Tema 2, 26-27, **Apuntes de Ampliación de Métodos Numéricos (2005)**, M. Marín

[4] Tema 2, 4-6, **Apuntes de Ampliación de Métodos Numéricos (2005)**, M. Marín

[5] Tema 2, 11-14, **Apuntes de Ampliación de Métodos Numéricos (2005)**, M. Marín