

# Actividad 7: Sistema de resortes acoplados

Melissa Matrecitos Avila

28 de Marzo de 2018

## 1 Introducción

El siguiente reporte corresponde a la actividad 7 del curso de Física Computacional 1, en la cual se estudió la modelación de fenómenos físicos, apoyados con Python, en especial con un sistema de resortes acoplados. Esto se logro mediante la aplicación de métodos numéricos, realizando las gráficas los resultados y comparando. cuando era posible, con la solución exacta.

EL reporte presenta una síntesis del artículo completo "Coupled spring equations" de Temple H. Fay y Sarah Duncan Graham junto con las secciones de código utilizadas para resolver el problema numericamente. También se incluye la comparación de los resultados numéricos con los exactos mediante la gráfica del error relativo. Por último se presentan las secciones de conclusión, bibliografía y apéndice.

## 2 Síntesis

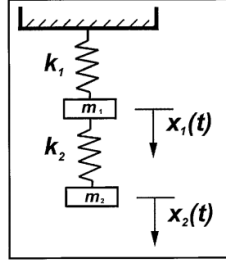
### 2.1 Introducción

El estudio clásico de las ecuaciones diferenciales está cambiando rápidamente, esto se debe en particular a la amplia disponibilidad de algoritmos numéricos de alta potencia y capacidades gráficas con un énfasis en las ecuaciones no lineales

En este artículo, se investiga el viejo problema de dos resortes y dos pesos unidos en serie, colgando del techo. Bajo la suposición de que las fuerzas de restauración se comportan de acuerdo con la Ley de Hooke, este problema de dos grados de libertad se modela mediante un par de ecuaciones diferenciales lineales de segundo orden acopladas. Al diferenciar y sustituir una ecuación por otra, el movimiento de cada peso se puede demostrar que está determinado por una ecuación diferencial lineal de cuarto orden.

Con las ecuaciones mencionadas se pueden estudiar algunos fenómenos más sobresalientes de este tipo de movimientos, como lo son la periodicidad, la amplitud, la fase, la sensibilidad a las condiciones iniciales y muchos otros conceptos, tan solo modificando los parámetros en el modelo.

## 2.2 El modelo de resorte acoplado



El modelo consiste en dos resortes y dos pesos. Un resorte con constante  $k_1$ , está unido al techo y un peso de masa  $m_1$  está unido al extremo inferior de este resorte. A este peso, se une un segundo resorte con una constante de resorte  $k_2$ . En la parte inferior de este segundo resorte, se adjunta un peso de masa  $m_2$ .

Considerando el sistema desde el equilibrio, medimos el desplazamiento del centro de masa de cada peso como una función del tiempo, y denotamos estas medidas por  $x_1(t)$  y  $x_2(t)$ , respectivamente.

### 2.2.1 Asumiendo la Ley de Hooke

Bajo la suposición de pequeñas oscilaciones, las fuerzas de restauración son de la forma  $-k_1 l_1$  y  $-k_2 l_2$  donde  $l_1$  y  $l_2$  son los alargamientos (o compresiones) de los dos resortes.

Dado que la masa superior está unida a ambos resortes, hay dos fuerzas de restauración que actúan sobre ella: una fuerza de restauración hacia arriba  $k_1 l_1$  ejercida por el alargamiento (o compresión)  $x_1$  del primer resorte; una fuerza hacia arriba  $k_2(x_2 - x_1)$  desde la resistencia del segundo muelle a ser alargada (o comprimida) en la cantidad  $x_2 - x_1$ . La segunda masa solo "siente" la fuerza de restauración desde el alargamiento (o compresión) del segundo resorte. Si asumimos que no hay fuerzas de amortiguamiento presentes, entonces la Ley de Newton implica que las dos ecuaciones que representan los movimientos de los dos pesos son:

$$m_1 \ddot{x}_1 = -k_1 x_1 - k_2(x_1 - x_2)$$

$$m_2 \ddot{x}_2 = -k_2(x_2 - x_1)$$

Para encontrar una ecuación para  $x_1$  y  $x_2$  que no dependan una de la otra, se resuelve la ecuación de  $x_2$  y se sustituye en la de  $x_1$ , obteniendo las ecuaciones:

$$\begin{aligned} m_1 m_2 x_1^{(4)} + (m_2 k_1 + k_2(m_1 + m_2)) \ddot{x}_1 + k_1 k_2 x_1 &= 0 \\ m_1 m_2 x_2^{(4)} + (m_2 k_1 + k_2(m_1 + m_2)) \ddot{x}_2 + k_1 k_2 x_2 &= 0 \end{aligned}$$

### 2.2.2 Algunos ejemplos con masas idénticas

Considerando un sistema normalizado con masas iguales, esto es  $m_1 = m_2 = 1$ , y asumiendo que no hay fricción ni fuerzas externas, la ecuación diferencial se reduce a:

$$m^{(4)} + (k_1 + 2k_2)m^2 + k_1 k_2 = 0$$

**Ejemplo 2.1:** Describe el movimiento para resortes con constantes  $k_1 = 6$  y  $k_2 = 4$  con condiciones iniciales  $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (1, 0, 2, 0)$ .

Resolviendo analíticamente encontramos que las soluciones para  $x_1$  y  $x_2$  son:

$$\begin{aligned}x_1(t) &= \cos \sqrt{2}t \\x_2(t) &= 2 \cos \sqrt{2}t\end{aligned}$$

El movimiento es sincronizado y las ondas se mueven fase una con la otra, teniendo el mismo periodo de movimiento pero con distintas amplitudes. Como el movimiento es periódico simple, las fases de  $x_1$  y  $x_2$  forman elipses.

A continuación se muestra el procedimiento correspondiente al método numérico utilizado en todos los ejemplos (variando solo las constantes indicadas), este se logra utilizando Python, obteniendo los datos y guardándolos en un archivo para después ser utilizados en las gráficas como se ha hecho en prácticas pasadas:

```
def vectorfield(w, t, p):
    """
    Defines the differential equations for the coupled spring-mass system.

    Arguments:
        w : vector of the state variables:
            w = [x1, y1, x2, y2]
        t : time
        p : vector of the parameters:
            p = [m1, m2, k1, k2, L1, L2, b1, b2]
    """
    x1, y1, x2, y2 = w
    m1, m2, k1, k2, L1, L2, b1, b2 = p

    # Create f = (x1', y1', x2', y2'):
    f = [y1,
        (-b1 * y1 - k1 * (x1 - L1) + k2 * (x2 - x1 - L2)) / m1,
        y2,
        (-b2 * y2 - k2 * (x2 - x1 - L2)) / m2]
    return f

# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
import numpy as np
import math

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 6.0
k2 = 4.0
# Natural Lengths
L1 = 0.0
L2 = 0.0
# Friction coefficients
b1 = 0.0
b2 = 0.0

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = 1.0
y1 = 0.0
x2 = 2.0
y2 = 0.0

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50.0
numpoints = 250

# Create the time samples for the output of the ODE solver.
# I use a large number of points, only because I want to make
# a plot of the solution that looks nice.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

# Pack up the parameters and initial conditions:
p = [m1, m2, k1, k2, L1, L2, b1, b2]
w0 = [x1, y1, x2, y2]

# Call the ODE solver.
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

with open('Ejemplo2.1.dat', 'w') as E21:
    # Print & save the solution.
    for ti, w1 in zip(t, wsol):
        print(ti, w1[0], w1[1], w1[2], w1[3], np.abs((w1[0] - (np.cos(np.sqrt(2)*ti)))) / (np.cos(np.sqrt(2)*ti))), np.abs((w1[2] - (2*np.cos(np.sqrt(2)*ti)))) / (2*np.cos(np.sqrt(2)*ti))), file=E21)
```

```
# Plot the solution that was generated

from numpy import loadtxt
from pylab import figure, plot, xlabel, grid, hold, legend, title, savefig
from matplotlib.font_manager import FontProperties
%matplotlib inline

t, x1, xy, x2, y2, e1, e2 = loadtxt('Ejemplo2_1.dat', unpack=True)

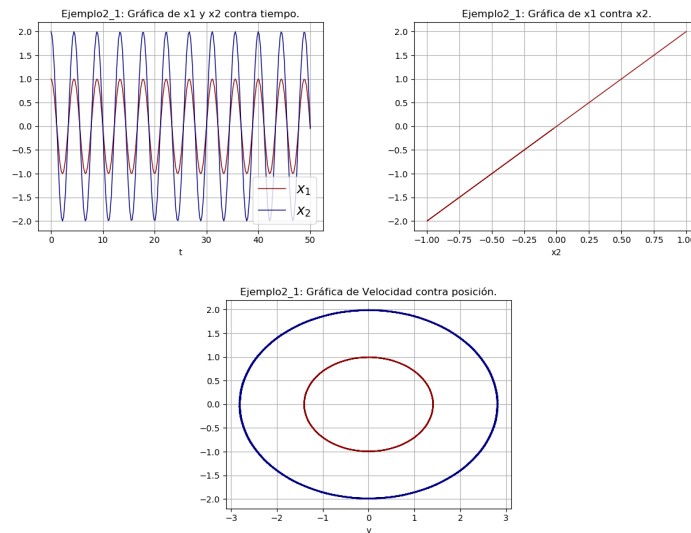
figure(1, figsize=(6, 4.5))

xlabel('t')
grid(True)
lw = 1

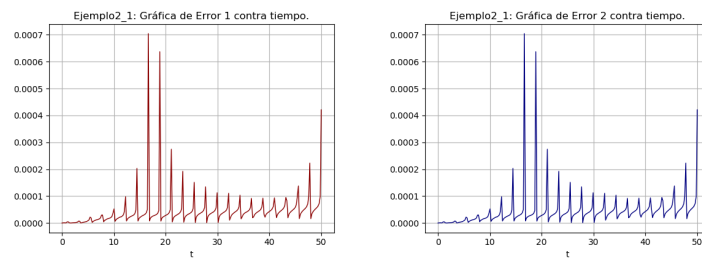
plot(t, x1, 'darkred', linewidth=lw)
plot(t, x2, 'navy', linewidth=lw)

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Ejemplo2_1: Gráfica de x1 y x2 contra tiempo.')
savefig('Ejemplo2_1Oscilaciones.png', dpi=100)
```

Las graficas obtenidas fueron:



Cómo el artículo presenta las soluciones analíticas, es posible calcular el error relativo para la posición de cada masa, obteniendo así las gráficas del error relativo contra el tiempo:



**Ejemplo 2.2:** Describe el movimiento para resortes con constantes  $k_1 = 6$  y  $k_2 = 4$  con condiciones iniciales  $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (-2, 0, 1, 0)$ .

Resolviendo analíticamente encontramos que las soluciones para  $x_1$  y  $x_2$  son:

$$x_1(t) = -2 \cos 2\sqrt{3}t$$

$$x_2(t) = \cos 2\sqrt{3}t$$

Al igual que el ejemplo anterior tienen el mismo periodo de movimiento y distintas amplitudes, solo que ahora mientras una se mueve en una dirección, la otra lo hace en sentido contrario. Así que realizando los cambios necesarios (como se muestra en la imagen) se obtuvieron las siguientes gráficas:

```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
import numpy as np
import math

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 6.0
k2 = 4.0
# Natural lengths
L1 = 0.0
L2 = 0.0
# Friction coefficients
b1 = 0.0
b2 = 0.0

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = -2.0
y1 = 0.0
x2 = 1.0
y2 = 0.0

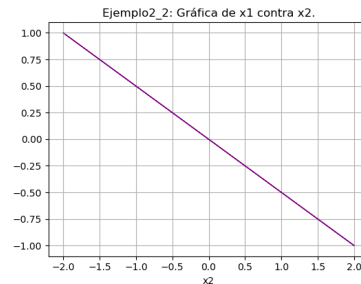
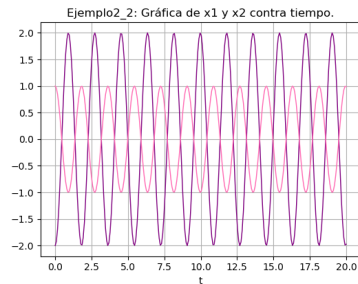
# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 20.0
numpoints = 250

# Create the time samples for the output of the ODE solver.
# I use a large number of points, only because I want to make
# a plot of the solution that looks nice.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

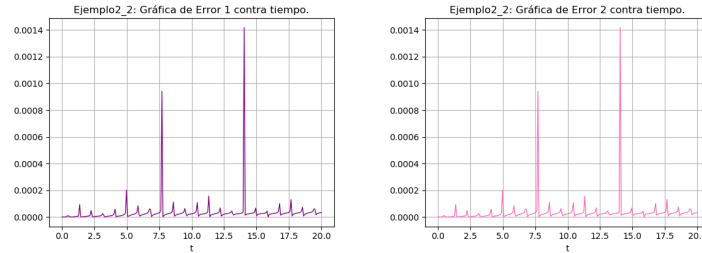
# Pack up the parameters and initial conditions:
p = [m1, m2, k1, k2, L1, L2, b1, b2]
w0 = [x1, y1, x2, y2]

# Call the ODE solver.
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

with open('Ejemplo2_2.dat', 'w') as E22:
    # Print & save the solution.
    for ti, w1 in zip(t, wsol):
        print(ti, w1[0], w1[1], w1[2], w1[3], np.abs((w1[0] - (-2*(np.cos(2*np.sqrt(3.0)*ti))))/(-2*(np.cos(2*np.sqrt(3.0)*ti))))), np.abs((w1[2] - (np.cos(2*np.sqrt(3.0)*ti)))/(np.cos(2*np.sqrt(3.0)*ti))))), file=E22)
```



Cómo en el ejemplo anterior, el artículo también presenta las soluciones analíticas, por lo que también es posible calcular el error relativo para la posición de cada masa, obteniendo así las gráficas del error relativo contra el tiempo:



**Ejemplo 2.3:** Describe el movimiento para resortes con constantes  $k_1 = 0.4$  y  $k_2 = 1.808$  con condiciones iniciales  $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (1/2, 0, -1/2, 7/10)$ .

En este ejemplo se nota como los valores de  $k_1$  y  $k_2$  determinan por completo el periodo y las condiciones iniciales solo afectan la amplitud y la fase de las soluciones. Este no cuenta con solución analítica, sin embargo aun se puede resolver con métodos numéricos:

```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 0.4
k2 = 1.808
# Natural lengths
L1 = 0.0
L2 = 0.0
# Friction coefficients
b1 = 0.0
b2 = 0.0

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = 0.5
y1 = 0.0
x2 = -0.5
y2 = 0.7

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50.0
numpoints = 500

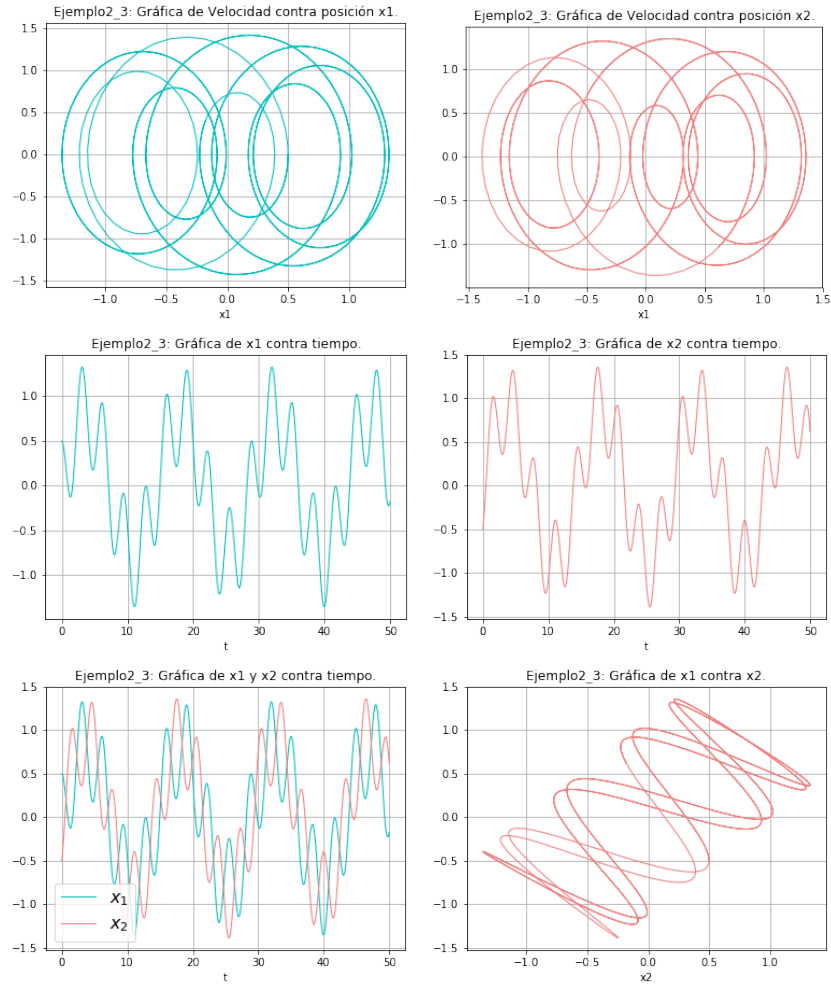
# Create the time samples for the output of the ODE solver.
# I use a large number of points, only because I want to make
# a plot of the solution that looks nice.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

# Pack up the parameters and initial conditions:
p = [m1, m2, k1, k2, L1, L2, b1, b2]
w0 = [x1, y1, x2, y2]

# Call the ODE solver.
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

with open('Ejemplo2_3.dat', 'w') as E23:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], w1[2], w1[3], file=E23)
```

Obteniendo las gráficas:



### 2.2.3 Amortiguamiento

EL tipo más común de amortiguamiento es el provocado por la viscosidad, donde la fuerza amortiguadora es proporcional a la velocidad. El amortiguamiento de la primera masa depende solo de su velocidad y no de la velocidad de la segunda masa y viceversa. Para agregar este fenómeno a las ecuaciones, se añaden los términos  $-\delta_1\dot{x}_1$  y  $-\delta_2\dot{x}_2$  a las ecuaciones correspondientes. Asumiendo que  $\delta_1$  y  $\delta_2$  son muy pequeños, el sistema queda:

$$\begin{aligned} m_1\ddot{x}_1 &= -\delta_1\dot{x}_1 - k_1x_1 - k_2(x_1 - x_2) \\ m_2\ddot{x}_2 &= -\delta_2\dot{x}_2 - k_2(x_2 - x_1) \end{aligned}$$

Realizando el mismo proceso para encontrar las nuevas ecuaciones para  $x_1$  y  $x_2$  que no dependan una de la otra, obtenemos:

$$m_1 m_2 x_2^{(4)} + (m_1 \delta_1 + m_2 \delta_2) \ddot{x}_1 + (m_2 k_1 + k_2(m_1 + m_2)) \ddot{x}_2 + k_1 k_2 x_2 = 0$$

**Ejemplo 2.4:** Asuma que  $m_1 = m_2 = 1$ . Describe el movimiento para resortes con constantes  $k_1 = 0.4$  y  $k_2 = 1.808$ , coeficientes de amortiguamiento  $\delta_1 = 0.1$  y  $\delta_2 = 0.2$  con condiciones iniciales  $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (1, 1/2, 2, 1/2)$ .

Debido al factor de amortiguamiento, la amplitud del movimiento disminuye conforme el tiempo avanza, lo cual se hace notorio en las gráficas de  $x_1$  contra tiempo y  $x_2$  contra tiempo.

```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 0.4
k2 = 1.808
# Natural lengths
L1 = 0.0
L2 = 0.0
# Friction coefficients
b1 = 0.1
b2 = 0.2

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = 1.0
y1 = 0.5
x2 = 2.0
y2 = 0.5

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50.0
numpoints = 250

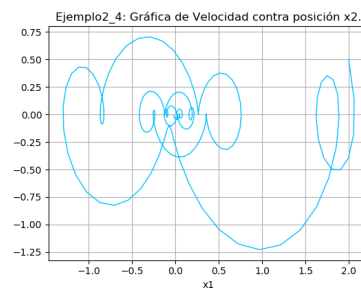
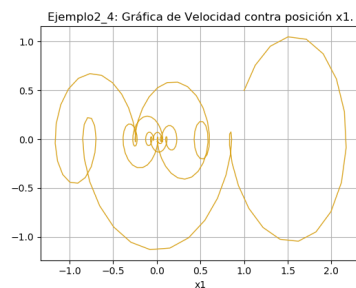
# Create the time samples for the output of the ODE solver.
# I use a large number of points, only because I want to make
# a plot of the solution that looks nice.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

# Pack up the parameters and initial conditions:
p = [m1, m2, k1, k2, L1, L2, b1, b2]
w0 = [x1, y1, x2, y2]

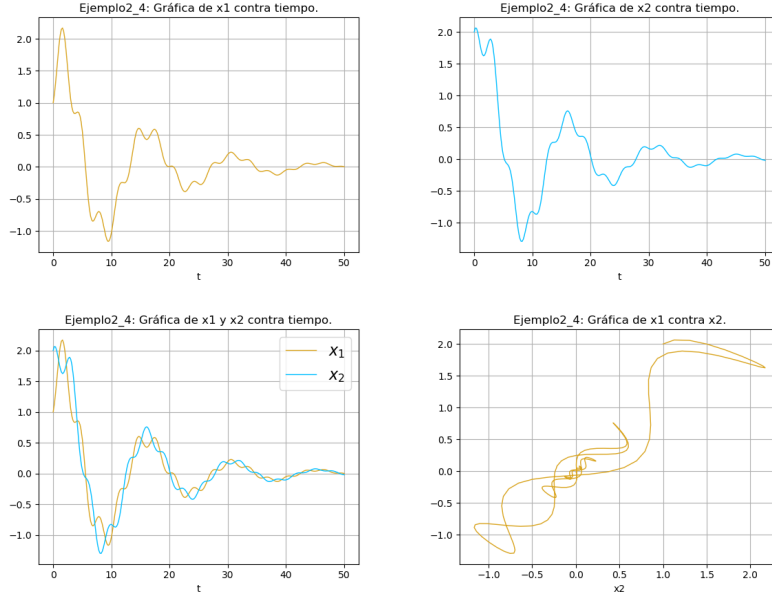
# Call the ODE solver.
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

with open('Ejemplo2_4.dat', 'w') as E24:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print(t1, w1[0], w1[1], w1[2], w1[3], file=E24)
```

Obteniendo las gráficas:







## 2.3 Añadiendo no linealidad

Si asumimos que las fuerzas restauradoras son no lineales y tienen la forma  $-kx + \mu x^3$ , entonces las ecuaciones quedan:

$$\begin{aligned} m_1 \ddot{x}_1 &= -\delta_1 \dot{x}_1 - k_1 x_1 + \mu_1 x_1^3 - k_2 (x_1 - x_2) + \mu_2 (x_2 - x_1)^3 \\ m_2 \ddot{x}_2 &= -\delta_2 \dot{x}_2 - k_2 (x_2 - x_1) + \mu_2 (x_2 - x_1)^3 \end{aligned}$$

La solución de este tipo de sistemas es mucho más complicada que la de los sistemas lineales. A continuación se muestran algunos ejemplos.

La función con la que se trabajaba se cambió para considerar el coeficiente no linealidad, quedando:

```
def vectorfield(w, t, p):
    """
    Defines the differential equations for the coupled spring-mass system.

    Arguments:
        w : vector of the state variables:
            w = [x1,y1,x2,y2]
        t : time
        p : vector of the parameters:
            p = [m1,m2,k1,k2,L1,L2,b1,b2,mi1,mi2]
    """
    x1, y1, x2, y2 = w
    m1, m2, k1, k2, L1, L2, b1, b2, mi1, mi2 = p

    # Create f = (x1',y1',x2',y2'):
    f = [y1,
        (-b1 * y1 - k1 * (x1 - L1) + mi1 * (x1 - L1)**3 + k2 * (x2 - x1 - L2) - mi2 * (x2 - x1 - L2)**3) / m1,
        y2,
        (-b2 * y2 - k2 * (x2 - x1 - L2) + mi2 * (x2 - x1 - L2)**3) / m2]
    return f
```

**Ejemplo 3.1:** Asuma que  $m_1 = m_2 = 1$ . Describe el movimiento para resortes con constantes  $k_1 = 0.4$  y  $k_2 = 1.808$ , coeficientes de amortiguamiento  $\delta_1 = 0$  y  $\delta_2 = 0$ , coeficientes de no linealidad  $\mu_1 = -1/6$  y  $\mu_2 = -1/10$ , y con condiciones iniciales  $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (1, 0, -1/2, 0)$ .

Cómo no hay amortiguamiento, el movimiento es oscilatorio y aparenta ser periódico. Por la no linealidad, el movimiento es muy sensible a condiciones iniciales.

```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
import numpy as np
import math

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 0.4
k2 = 1.808
# Natural Lengths
L1 = 0.0
L2 = 0.0
# Friction coefficients
b1 = 0.0
b2 = 0.0
# Nonlinear coefficients
mi1 = -1/6
mi2 = -1/10

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = 1.0
y1 = 0.0
x2 = -0.5
y2 = 0.0

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50.0
numpoints = 2000

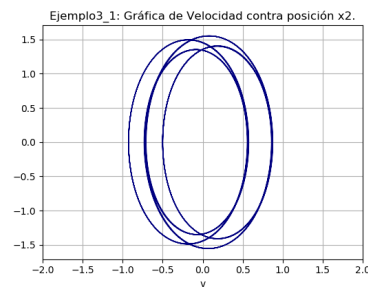
# Create the time samples for the output of the ODE solver.
# I use a large number of points, only because I want to make
# a plot of the solution that looks nice.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

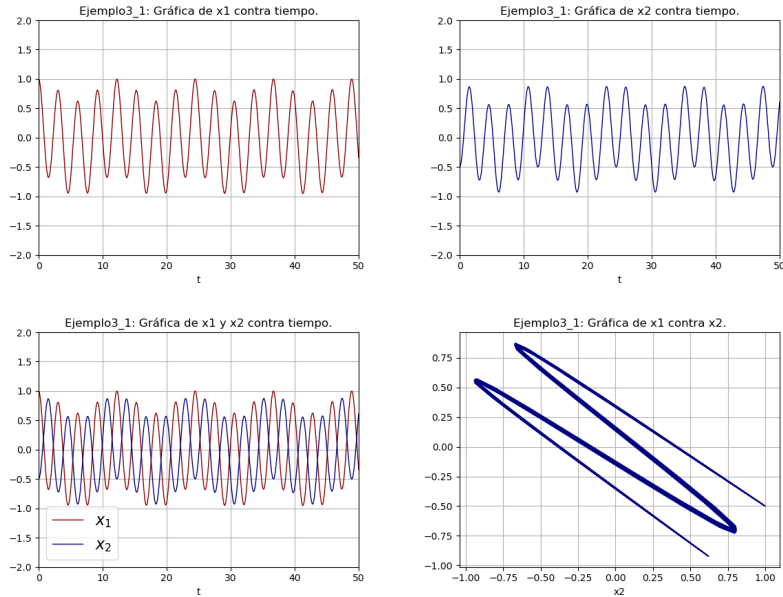
# Pack up the parameters and initial conditions:
p = [m1, m2, k1, k2, L1, L2, b1, b2, mi1, mi2]
w0 = [x1, y1, x2, y2]

# Call the ODE solver.
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

with open('Ejemplo3_1.dat', 'w') as E31:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print(t1, w1[0], w1[1], w1[2], w1[3], file=E31)
```

Obteniendo las gráficas:





**Ejemplo 3.2:** Asuma que  $m_1 = m_2 = 1$ . Describe el movimiento para resortes con constantes  $k_1 = 0.4$  y  $k_2 = 1.808$ , coeficientes de amortiguamiento  $\delta_1 = 0$  y  $\delta_2 = 0$ , coeficientes de no linealidad  $\mu_1 = -1/6$  y  $\mu_2 = -1/10$ , y con condiciones iniciales  $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (-0.5, 1/2, 3.001, 5.9)$ .

```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
import numpy as np
import math

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 0.4
k2 = 1.808
# Natural lengths
L1 = 0.0
L2 = 0.0
# Friction coefficients
b1 = 0.0
b2 = 0.0
# Nonlinear coefficients
mu1 = -1/6
mu2 = -1/10

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = -0.5
y1 = 0.5
x2 = 3.001
y2 = 5.9

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stopline = 50
numpoints = 2000

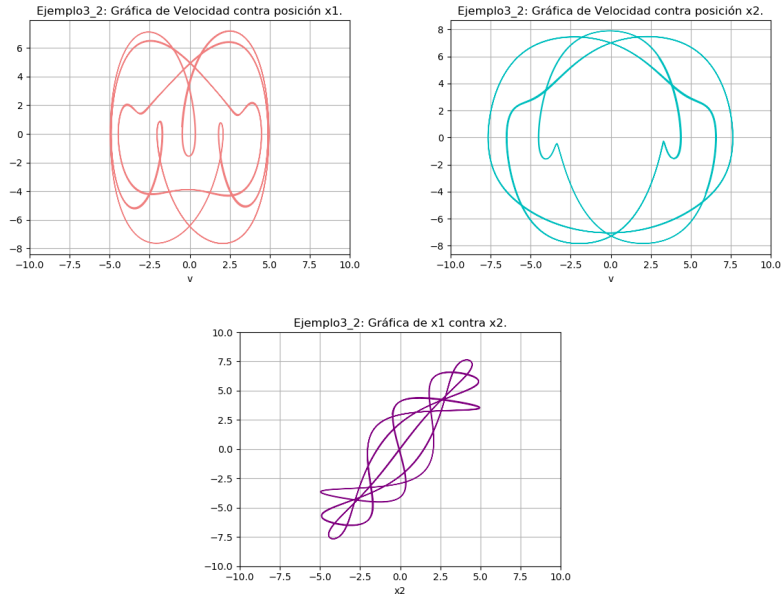
# Create the time samples for the output of the ODE solver.
# I use a large number of points, only because I want to make
# a plot of the solution that looks nice.
t = [stopline * float(i) / (numpoints - 1) for i in range(numpoints)]

# Pack up the parameters and initial conditions:
p = [m1, m2, k1, k2, L1, L2, b1, b2, mu1, mu2]
w0 = [x1, y1, x2, y2]

# Call the ODE solver.
wsol = odeint(vectorfield, w0, t, args=(p,))
atol=abserr, rtol=relerr

with open('Ejemplo3_2.dat', 'w') as E32:
    # Print & save the solution.
    for ti, w1 in zip(t, wsol):
        print(ti, w1[0], w1[1], w1[2], w1[3], file=E32)
```

Obteniendo las gráficas:



**Ejemplo 3.2:** Asuma que  $m_1 = m_2 = 1$ . Describe el movimiento para resortes con constantes  $k_1 = 0.4$  y  $k_2 = 1.808$ , coeficientes de amortiguamiento  $\delta_1 = 0$  y  $\delta_2 = 0$ , coeficientes de no linealidad  $\mu_1 = -1/6$  y  $\mu_2 = -1/10$ , y con condiciones iniciales  $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (-0.6, 1/2, 3.001, 5.9)$ .

```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
import numpy as np
import math

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 0.4
k2 = 1.808
# Natural lengths
L1 = 0.0
L2 = 0.0
# Friction coefficients
b1 = 0.0
b2 = 0.0
# Nonlinear coefficients
mu1 = -1/6
mu2 = -1/10

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = -0.6
y1 = 0.5
x2 = 3.001
y2 = 5.9

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stopime = 200
numpoints = 4000

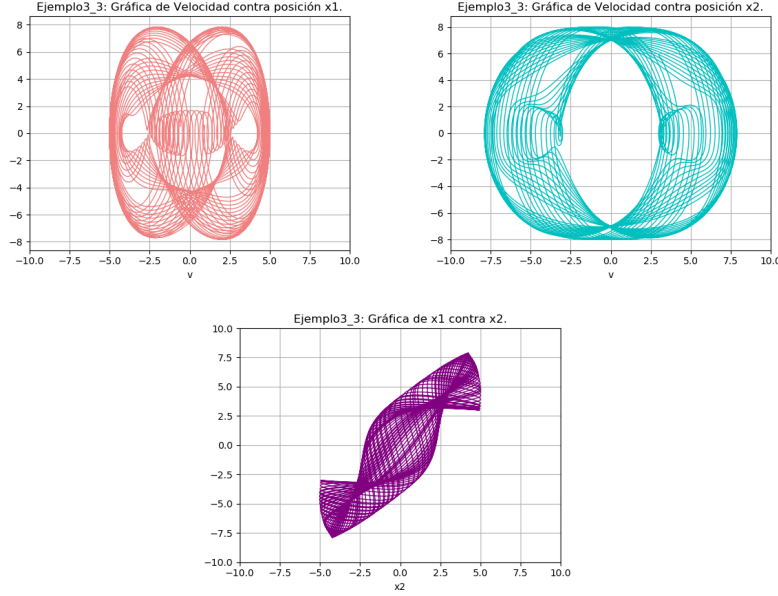
# Create the time samples for the output of the ODE solver.
# I use a large number of points, only because I want to make
# a plot of the solution that looks nice.
t = [stopime * float(i) / (numpoints - 1) for i in range(numpoints)]

# Pack up the parameters and initial conditions:
p = [m1, m2, k1, k2, L1, L2, b1, b2, mu1, mu2]
w0 = [x1, y1, x2, y2]

# Call the ODE solver.
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

with open('Ejemplo3_2.dat', 'w') as E33:
    # Print & save the solution.
    for ti, w1 in zip(t, wsol):
        print(ti, w1[0], w1[1], w1[2], w1[3], file=E33)
```

Obteniendo las gráficas:



## 2.4 Añadiendo forzamiento

Se puede añadir una fuerza externa de una manera muy sencilla al modelo, siendo o no igual para ambas masas. Suponiendo que se aplica una fuerza senoidal con forma  $F \cos(wt)$ , las ecuaciones quedan:

$$\begin{aligned} m_1 \ddot{x}_1 &= -\delta_1 \dot{x}_1 - k_1 x_1 + \mu_1 x_1^3 - k_2(x_1 - x_2) + \mu_2(x_2 - x_1)^3 + F_1 \cos(w_1 t) \\ m_2 \ddot{x}_2 &= -\delta_2 \dot{x}_2 - k_2(x_2 - x_1) + \mu_2(x_2 - x_1)^3 + F_2 \cos(w_2 t) \end{aligned}$$

El rango de movimientos para un sistema no lineal es bastante amplio. Las condiciones para que este movimiento ocurra son muy difíciles de especificar, por lo que se muestra un ejemplo:

**Ejemplo 4.1:** Asuma que  $m_1 = m_2 = 1$ . Describe el movimiento para resortes con constantes  $k_1 = 2/5$  y  $k_2 = 1$ , coeficientes de amortiguamiento  $\delta_1 = 1/10$  y  $\delta_2 = 1/5$ , coeficientes de no linealidad  $\mu_1 = 1/6$  y  $\mu_2 = 1/10$ , fuerzas de amplitud  $F_1 = 1/3$  y  $F_2 = 1/5$ , frecuencias de forzamiento  $w_1 = 1$  y  $w_2 = 3/5$  y con condiciones iniciales  $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (0.7, 0, 0.1, 0)$ .

Cómo hay amortiguamiento, el movimiento es muy inestable para valores pequeños de  $t$ , pero conforme  $t$  crezca, el movimiento pareciera constante. La función a utilizar se cambió a la siguiente:

```

import numpy as np
def vectorfield(w, t, p):
    """
    Defines the differential equations for the coupled spring-mass system.

    Arguments:
        w : vector of the state variables:
            w = [x1,y1,x2,y2]
        t : time
        p : vector of the parameters:
            p = [m1,m2,k1,k2,L1,L2,b1,b2,mi1,mi2,F1, F2, w1, w2]
    """
    x1, y1, x2, y2 = w
    m1, m2, k1, k2, L1, L2, b1, b2, mi1, mi2, F1, F2, w1, w2 = p

    # Create f = (x1',y1',x2',y2'):
    f = [y1,
         (-b1 * y1 - k1 * (x1 - L1) + mi1 * (x1 - L1)**3 + k2 * (x2 - x1 - L2) - mi2 * (x2 - x1 - L2)**3 +
         F1*np.cos(w1*t)) / m1,
         y2,
         (-b2 * y2 - k2 * (x2 - x1 - L2) + mi2 * (x2 - x1 - L2)**3 + F2*np.cos(w2*t)) / m2]
    return f

```

```

# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
import numpy as np
import math

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 2/5
k2 = 1
# Natural Lengths
L1 = 0.0
L2 = 0.0
# Friction coefficients
b1 = 1/10
b2 = 1/5
# Nonlinear coefficients
mi1=1/6
mi2=1/10

#Fuerzas iniciales
F1=1/3
F2=1/5

#Frecuencias iniciales
w1=1
w2=3/5

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = 0.7
y1 = 0
x2 = 0.1
y2 = 0

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 250
numpoints = 2000

# Create the time samples for the output of the ODE solver.
# I use a large number of points, only because I want to make
# a plot of the solution that looks nice.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

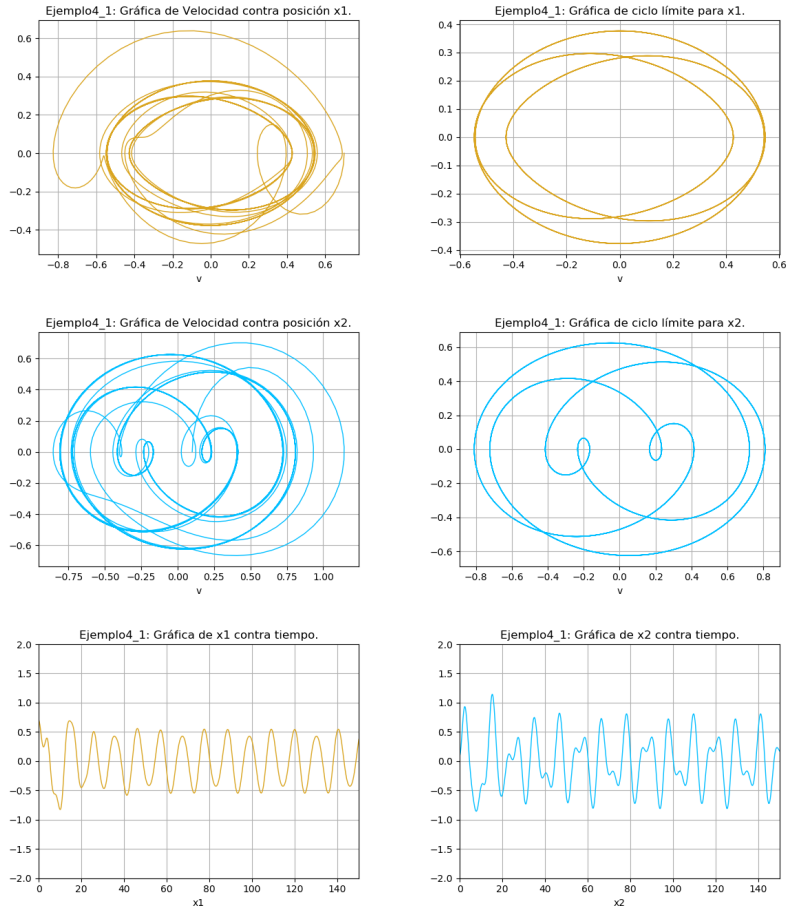
# Pack up the parameters and initial conditions:
p = [m1, m2, k1, k2, L1, L2, b1, b2, mi1, mi2, F1, F2, w1, w2]
w0 = [x1, y1, x2, y2]

# Call the ODE solver.
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

with open('Ejemplo4_1.dat', 'w') as E41:
    # Print & save the solution.
    for ti, w1 in zip(t, wsol):
        print(ti, w1[0], w1[1], w1[2], w1[3], file=E41)

```

Obteniendo las gráficas:



### 3 Conclusión

Con esta actividad se completó el resumen de todo el artículo, mostrando en las ultimas secciones movimientos que nunca me había imaginado, pero lo que más me sorprendió fue la gran herramienta que fue Python para resolver los ejemplos que ahí venían.

Aun que incluso se mencionaba que algunos programas numéricos no podrían realizar buenas aproximaciones a los modelos mostrados, las gráficas resultantes fueron idénticas a los del ejemplo.

Y obviamente como en la práctica pasada, el ambiente de Jupyter Lab mejoró bastante la realización de esta actividad.

## 4 Bibliografía

- H. FAY, T., & DUNCAN GRAHAM, S. (2003). Coupled spring equations (pp. 65-79). Int. J. Math. Educ. Sci. Technol. Recuperado de [http://math.oregonstate.edu/~gibsonn/Teaching/MTH323-010S15/Supplements/coupled\\_spring.pdf](http://math.oregonstate.edu/~gibsonn/Teaching/MTH323-010S15/Supplements/coupled_spring.pdf)

## 5 Apéndice

1. ¿Qué más te llama la atención de la actividad completa? ¿Que se te hizo menos interesante?

Lo que más me gustó fue que ahora en lugar de solo modificar las condiciones iniciales, se tuvo que cambiar varias veces la función a utilizar para poder modelar la situación.

2. ¿De un sistema de masas acopladas como se trabaja en esta actividad, hubieras pensado que abre toda una nueva área de fenómenos no lineales?

Sabía que habían algunos fenómenos interesantes, pero nunca los había trabajado ya que en otros cursos se despreciaban los coeficientes que provocaban estos fenómenos.

3. ¿Qué propondrías para mejorar esta actividad? ¿Te ha parecido interesante este reto? Me parece que está muy bien así, ya que relacionamos temas de otras materias con la programación, encontrando nuevos métodos par abordar los problemas.

4. ¿Quisieras estudiar mas este tipo de fenómenos no lineales? Sí, por que en otros cursos no tenemos las herramientas para poder profundizar en el tema.