"El saber de mis hijos
hará mi grandeza"

Universidad De Sonora

División de Ciencias Exactas y Naturales

Licenciatura En Física

Física Computacional I

Reporte de Actividad 7

"Sistema de Resortes Acoplados"

César Omar Ramírez Álvarez

Profr. Carlos Lizárraga Celaya

Hermosillo, Sonora Marzo 24 de 2018

Continuación...

El presente informe es producto de la séptima práctica de Física Computacional I, en la que continuaremos con los ejemplos 3.1, 3.2, 3,.3 y 4.1 del artículo "Coupled Spring Equations" de los autores Fay y Graham.

Añadiendo No Linealidad

Ahora las fuerzas restauradoras de los resortes no obedecen la Ley de Hooke, por ende, en este caso se debe modificar el modelo que se tiene. Contaremos con una nueva constante "μ". El nuevo modelo queda de la siguiente manera:

$$m_1\ddot{x}_1 = -\delta\dot{x}_1 - k_1x_1 - k_2(x_1 - x_2) + \mu_1(x_1 - x_2)^3$$
$$m_2\ddot{x}_2 = -\delta\dot{x}_2 - k_2(x_2 - x_1) + \mu_2(x_2 - x_1)^3$$

El rango de movimientos para el modelo no lineal es mucho más complicado que para el modelo lineal. Además, surgen preguntas de precisión al resolver estas ecuaciones. No se puede esperar que ningún solucionador numérico se mantenga preciso durante largos intervalos de tiempo. A continuación se presentan alguno ejemplos.

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

A continuación, se presenta la sección de código utilizada para encontrar numéricamente los resultados con Python en Jupyter Lab. (La primera imagen es la misma para los ejemplos 3.1, 3.2 y 3.3)

```
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,c1,c2]
    """
    x1, y1, x2, y2 = w
    m1, m2, k1, k2, L1, L2, b1, b2, c1, c2 = p

    # Create f = (x1',y1',x2',y2'):
    f = [y1,
         (-b1 * y1 - k1 * (x1 - L1) + k2 * (x2 - x1 - L2) + c1 * (x1)**3 + c2 * (x1 - x2)**3) / m1,
         y2,
         (-b2 * y2 - k2 * (x2 - x1 - L2) + c2 * (x2 - x1)**3) / m2]
    return f
```

```
# 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
#Las b's
c1 = -1/6
c2 =-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 = -1/2
y2 = 0.0

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50.0
numpoints = 1250

# 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, c1, c2]
w0 = [x1, y1, x2, y2]

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

with open('Ejercicio3.1.dat', 'w') as f:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], w1[2], w1[3], file=f)
```

```
# 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
import matplotlib.pyplot as plt
%matplotlib inline

t, x1, xy, x2, y2 = loadtxt('Ejercicio3.1.dat', unpack=True)

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

grid(True)

lw = 1
```

```
plot(x1, xy, 'b', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)
plt.xlim([-2,2])

title('Phase Plot for x1')
savefig('G3.1a.png', dpi=100)
```

```
plot(x2, y2, 'g', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)
plt.xlim([-2,2])

title('Phase Plot for x2')
savefig('G3.1b.png', dpi=100)
```

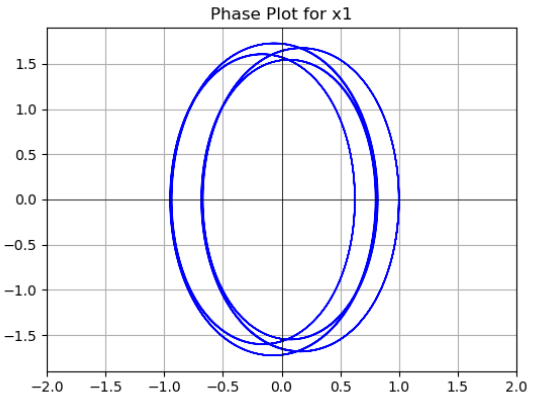
```
plot(t, x1, 'b', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.ylim([-2,2])

title('Plot for x1')
savefig('G3.1c.png', dpi=100)
```

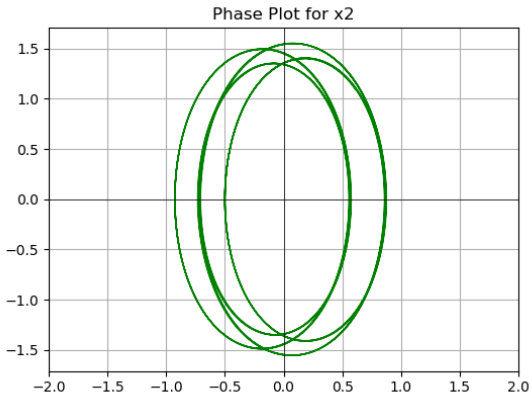
```
plot(t, x2, 'g', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.ylim([-2,2])

title('Plot for x2')
savefig('G3.1d.png', dpi=100)
```

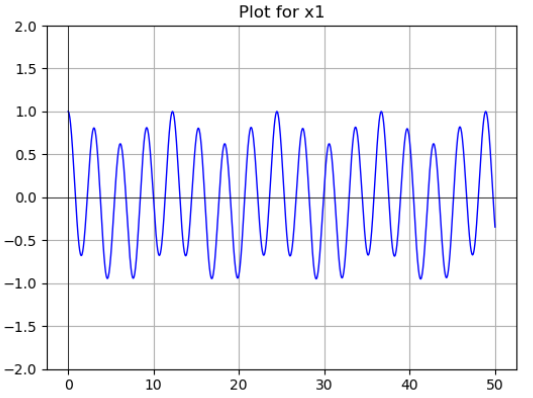
```
plot(t, x2, 'b', linewidth=lw)
plot(t, x1, 'g', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.ylim([-2,2])

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Plot of x1 and x2')
savefig('G3.1e.png', dpi=100)
```

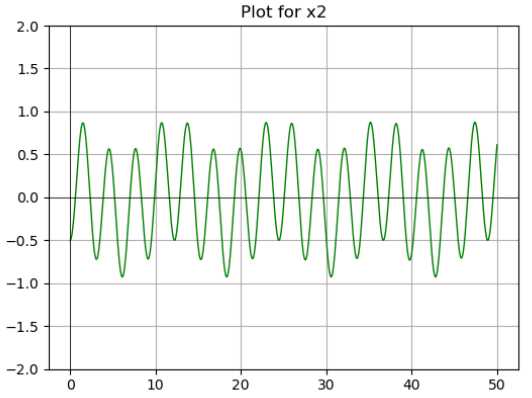
```
plot(x1, x2, 'r', linewidth=lw)
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.ylim([-1.5,1.5])
plt.xlim([-1.5,1.5])

title('Plot of x1 versus x2')
savefig('G3.1f.png', dpi=100)
```

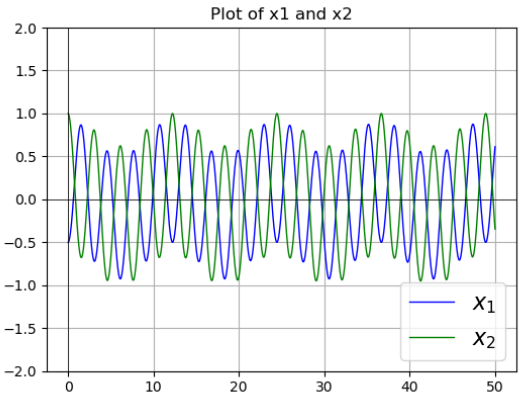
Las gráficas que se obtuvieron son:

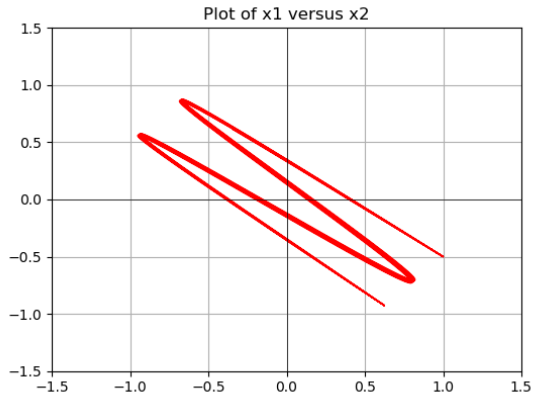












De lo anterior es notorio que debido a la nolinealidad, el modelo parece ser más sensible a las condiciones iniciales, es decir, como no tenemos un amortiguamiento las oscilaciones se tornan casi periódicas, así al graficar x_1 con x_2 contra el tiempo, sus movimientos parecen estar fuera de fase.

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

A continuación, se presenta la sección de código utilizada para encontrar numéricamente los resultados con Python en Jupyter Lab.

```
# 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
#Las b's
c1 = -1/6
c2 = -1/10
# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = -0.5
y1 = 1/2
x2 = 3.001
y2 = 5.9
```

```
# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50.0
numpoints = 1250

# 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, c1, c2]
w0 = [x1, y1, x2, y2]

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

with open('Ejercicio3.2.dat', 'w') as f:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], w1[2], w1[3], file=f)
```

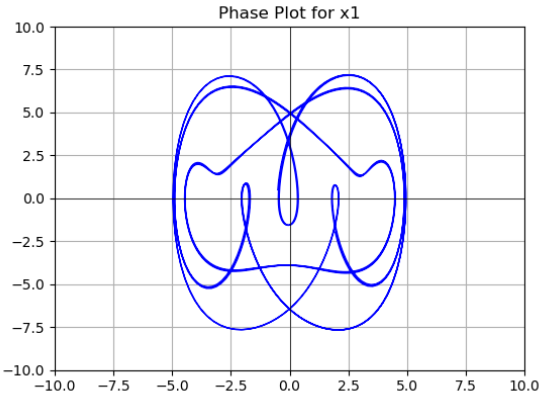
```
# Plot the solution that was generated

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

t, x1, xy, x2, y2 = loadtxt('Ejercicio3.2.dat', unpack=True)

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

grid(True)

lw = 1
```

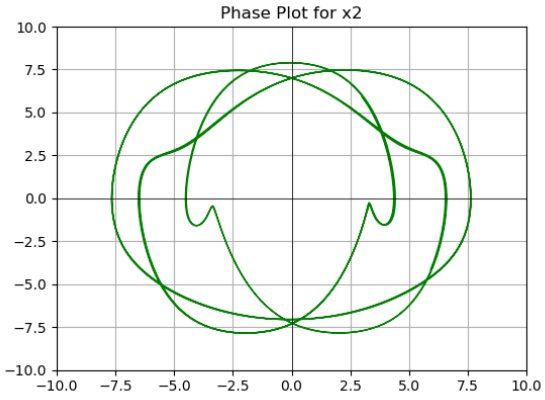
```
plot(x1, xy, 'b', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.xlim([-10,10])
plt.ylim([-10,10])

title('Phase Plot for x1')
savefig('G3.2a.png', dpi=100)
```

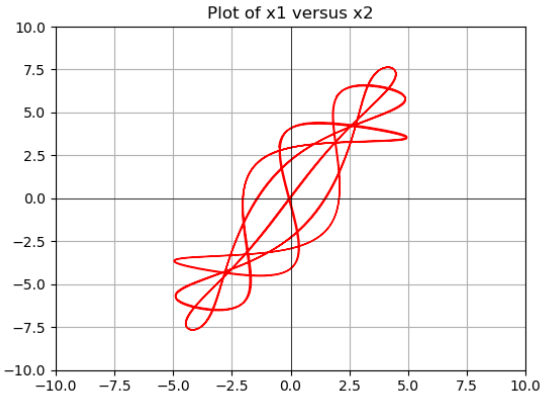
```
plot(x2, y2, 'g', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.xlim([-10,10])
plt.ylim([-10,10])

title('Phase Plot for x2')
savefig('G3.2b.png', dpi=100)
```

```
plot(x1, x2, 'r', linewidth=lw)
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.ylim([-10,10])
plt.xlim([-10,10])

title('Plot of x1 versus x2')
savefig('G3.2c.png', dpi=100)
```

Las gráficas que se obtuvieron son:







Observamos que con un simple cambio en las condiciones iniciales del modelo se muestra un gran cambio en las gráficas (es lo que cambia respecto al ejemplo anterior).

3.3 Asume $m_1 = m_2 = 1$***. Describe el movimiento para un sistema de resortes con*** $k_1 = 0.4$ ***y*** $k_2 = 1.808$***, con coeficientes de amortiguamiento*** $\delta_1 = 0$ ***y*** $\delta_2 = 0$***, coeficientes de no lineales de*** $\mu_1 = -1/6$ ***y*** $\mu_2 = -1/10$ ***con condiciones iniciales de*** $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0))$=(−0.6,1/2,3.001,5.9)***.***

Ahora observaremos el comportamiento en el que hay una diferencia de 0.1 en las condiciones inicales respecto al ejemplo anterior, es de esperar que se observe un gran cambio en las gráficas respecto a las anteriores.

A continuación, se presenta la sección de código utilizada para encontrar numéricamente los resultados con Python en Jupyter Lab.

```
# 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
#Las b's
c1 = -1/6
c2 = -1/10
# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = -0.6
y1 = 1/2
x2 = 3.001
y2 = 5.9

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 200.0
numpoints = 2500

# 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, c1, c2]
w0 = [x1, y1, x2, y2]

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

with open('Ejercicio3.3.dat', 'w') as f:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], w1[2], w1[3], file=f)
```

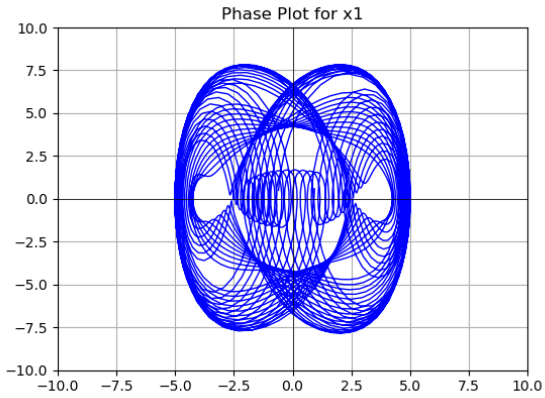
```
# Plot the solution that was generated

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

t, x1, xy, x2, y2 = loadtxt('Ejercicio3.3.dat', unpack=True)

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

grid(True)

lw = 1
```

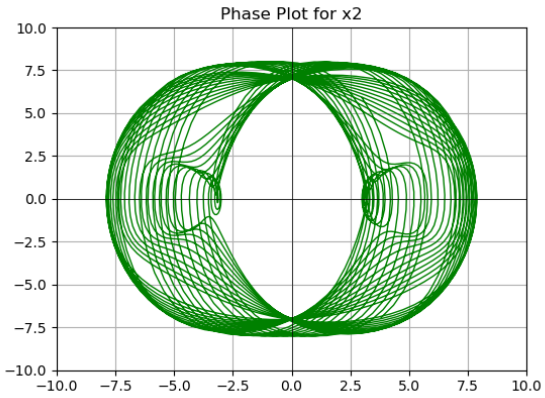
```
plot(x1, xy, 'b', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.xlim([-10,10])
plt.ylim([-10,10])

title('Phase Plot for x1')
savefig('G3.3a.png', dpi=100)
```

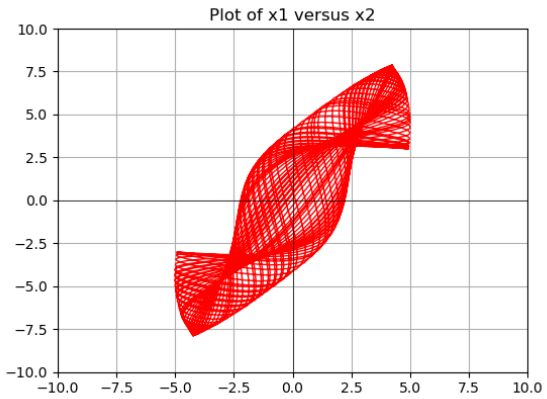
```
plot(x2, y2, 'g', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.xlim([-10,10])
plt.ylim([-10,10])

title('Phase Plot for x2')
savefig('G3.3b.png', dpi=100)
```

```
plot(x1, x2, 'r', linewidth=lw)
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.ylim([-10,10])
plt.xlim([-10,10])

title('Plot of x1 versus x2')
savefig('G3.3c.png', dpi=100)
```

Las gráficas que se obtuvieron son:







Como lo habiamos predecido, un breve cambio en las condiciones iniciales genera mucha diferencia en los productos de los sistemas no lineales.

Añadiendo Forzamiento

Es una cuestión simple agregar forzamiento externo al modelo. De hecho, podemos manejar cada masa de manera diferente. Supongamos un forzamiento senoidal simple. Entonces el modelo se convierte en:

$$m_1\ddot{x}_1 = -\delta\dot{x}_1 - k_1x_1 - k_2(x_1 - x_2) + \mu_1(x_1 - x_2)^3 + F_1cos(\omega_1 t)$$
$$m_2\ddot{x}_2 = -\delta\dot{x}_2 - k_2(x_2 - x_1) + \mu_2(x_2 - x_1)^3 + F_2cos(\omega_2 t)$$

El rango de movimientos para modelos forzados no lineales es bastante ampliO. Podemos esperar encontrar soluciones limitadas y sin límites (resonancia no lineal), soluciones periódicas que comparten el período con el forzamiento (llamadas soluciones armónicas) y las soluciones que son periódicas del período un múltiplo del período de conducción (llamadas soluciones subarmónicas), y las soluciones periódicas de estado Estacionario (ciclos límite en el plano de fase). Las condiciones bajo las cuales ocurren estos movimientos no son de ninguna manera fáciles de expresar. Concluimos con un simple ejemplo forzado.

4.1 Asume $m_1 = m_2 = 1$***. Describe el movimiento para un sistema de resortes con*** $k_1 = 2/5$ ***y*** $k_2 = 1$***, con coeficientes de amortiguamiento*** $\delta_1 = 1/10$ ***y*** $\delta_2 = 1/5$***, coeficientes de no lineales de*** $\mu_1 = 1/6$ ***y*** $\mu_2 = 1/10$***, fuerzas de amplitud de*** $F_1 = 1/3$ ***y*** $F_2 = 3/5$ ***y frecuencias de amplitud de*** $\omega_1 = 1$ ***y*** $\omega_2 = 3/5$ ***con condiciones iniciales de*** $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0))$=(0.7,0,0.1,0)***.***

```
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,c1,c2,f1,f2,w1,w2]
    """
    x1, y1, x2, y2 = w
    m1, m2, k1, k2, L1, L2, b1, b2, c1, c2, f1, f2, w1, w2 = p

    # Create f = (x1',y1',x2',y2'):
    f = [y1,
         (-b1 * y1 - k1 * (x1 - L1) + k2 * (x2 - x1 - L2) + c1 * (x1)**3 + c2 * (x1 - x2)**3 + f1 * np.cos(w1*t)) / m1,
         y2,
         (-b2 * y2 - k2 * (x2 - x1 - L2) + c2 * (x2 - x1)**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

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 2/5
k2 = 1.0
# Natural lengths
L1 = 0.0
L2 = 0.0
# Friction coefficients
b1 = 1/10
b2 = 1/5
#Las b's
c1 = 1/6
c2 = 1/10
#Las F's
f1 = 1/3
f2 = 1/5
#Las w´s
w1 = 1.0
w2 = 3/5
# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = 0.7
y1 = 0.0
x2 = 0.1
y2 = 0.0

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 200.0
numpoints = 2500

# 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, c1, c2, 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('Ejercicio4.1_0.dat', 'w') as f:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], w1[2], w1[3], file=f)
```

```
# 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
import matplotlib.pyplot as plt
%matplotlib inline

t, x1, xy, x2, y2 = loadtxt('Ejercicio4.1_0.dat', unpack=True)

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

grid(True)

lw = 1
```

```
plot(x1, xy, 'b', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.xlim([-1,1])
plt.ylim([-1,1])

title('Phase Plot for x1')
savefig('G4.1a.png', dpi=100)
```

```
# 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
import matplotlib.pyplot as plt
%matplotlib inline

t, x1, xy, x2, y2 = loadtxt('Ejercicio4.1_1.dat', unpack=True, skiprows=800)

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

grid(True)

lw = 1
```

```
plot(x1, xy, 'b', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.xlim([-0.6,0.6])
plt.ylim([-0.6,0.6])

title('Limit Cycle for x1')
savefig('G4.1b.png', dpi=100)
```

```
plot(x2, y2, 'g', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.xlim([-1.5,1.5])
plt.ylim([-1.5,1.5])

title('Phase Plot for x2')
savefig('G4.1c.png', dpi=100)
```

```
plot(x2, y2, 'g', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.xlim([-1.0,1.0])
plt.ylim([-1.0,1.0])

title('Limit Cycle for x2')
savefig('G4.1d.png', dpi=100)
```

```
plot(t, x1, 'b', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.ylim([-2,2])
plt.xlim([-1,140])

title('Plot for x1')
savefig('G4.1e.png', dpi=100)
```

```
plot(t, x2, 'g', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.ylim([-2,2])
plt.xlim([-1,140])

title('Plot for x2')
savefig('G4.1f.png', dpi=100)
```

```
# 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
import matplotlib.pyplot as plt
%matplotlib inline

t, x1, xy, x2, y2 = loadtxt('Ejercicio4.1_2.dat', unpack=True, skiprows=160)

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

grid(True)

lw = 1
```

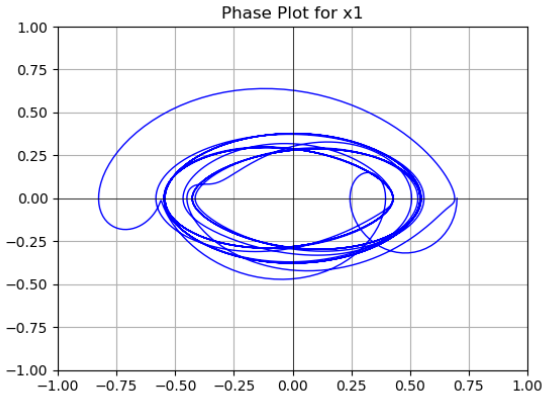
```
plot(t, x2, 'b', linewidth=lw)
plot(t, x1, 'g', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.ylim([-2,2])
plt.xlim([110,170])

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Plot of x1 and x2')
savefig('G4.1g.png', dpi=100)
```

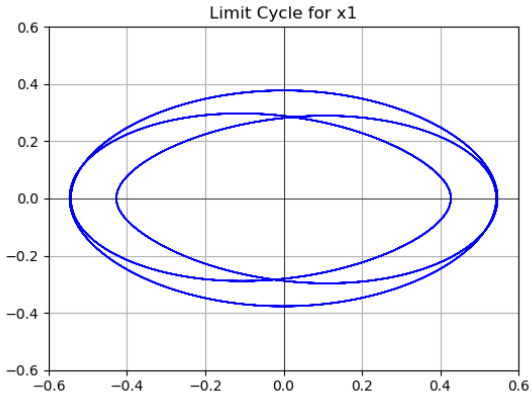
```
# Plot the solution that was generated

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

t, x1, xy, x2, y2 = loadtxt('Ejercicio4.1_2.dat', unpack=True, skiprows=800)

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

grid(True)

lw = 1
```

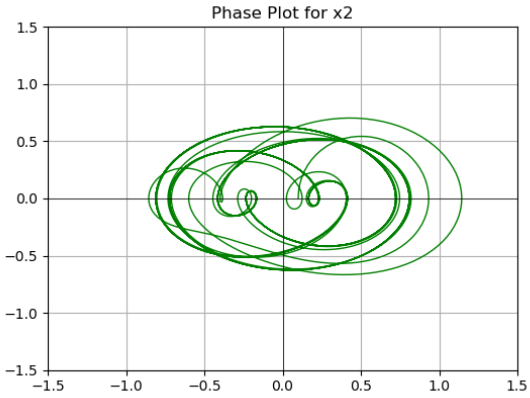
```
plot(x1, x2, 'r', linewidth=lw)

plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

plt.ylim([-1.0,1.0])
plt.xlim([-1.0,1.0])

title('Plot of x1 versus x2')
savefig('G4.1h.png', dpi=100)
```

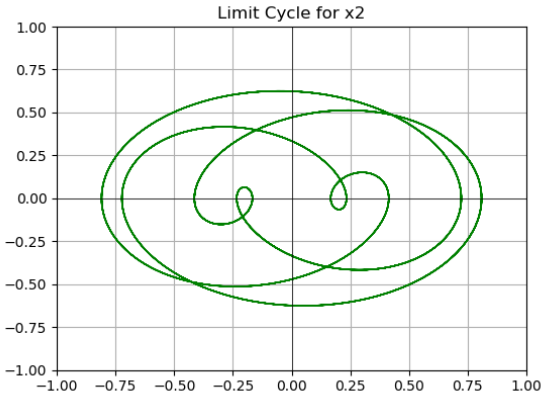
Como en este caso se hace uso de ciclos límite, fue necesario crear tres archivos con distitnto número de puntos y stoptime. En los códigos para generar las gráficas se nota cual de los archivos se usó.

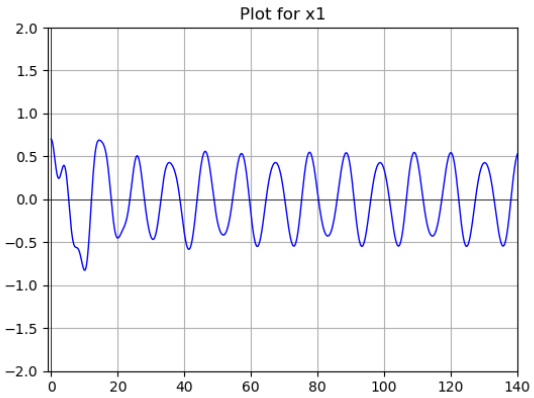
Las gráficas que se obtuvieron son:

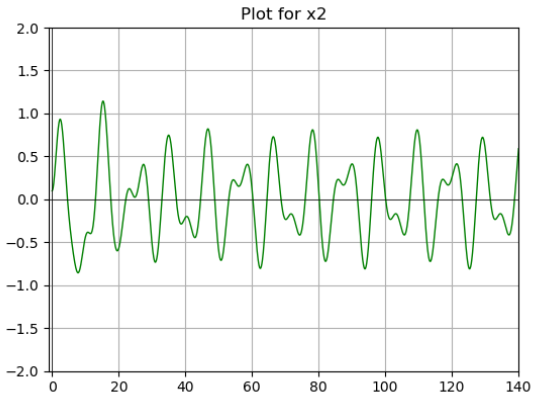


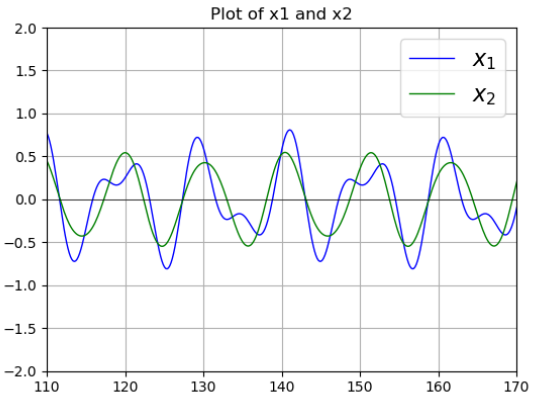


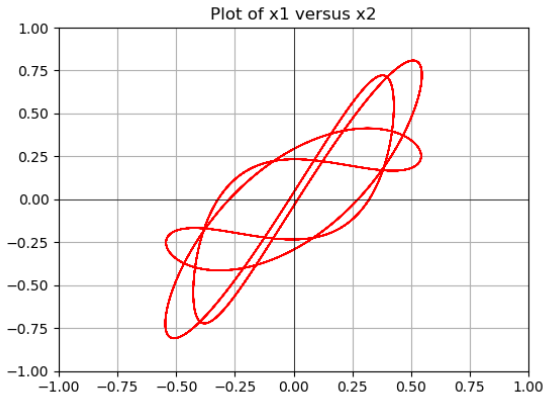












Con la existencia de un coeficiente de amortiguamiento, para valores de tiempo pequeños hay movimiento variable, pero para valores grandes del tiempo se torna un movimiento constante.Notorio en las graficas de ciclo límite.

Conclusión

Hemos desarrollado un modelo simple para dos resortes acoplados, hemos examinado tanto el caso lineal como una forma posible para el caso no lineal, y hemos incluido ejemplos de movimiento libre, movimiento amortiguado y movimiento forzado.

El modelo tiene muchas características que permiten la introducción significativa de muchos conceptos que incluyen: precisión de algoritmos numéricos, dependencia de parámetros y condiciones iniciales, fase y sincronización, periodicidad, tiempos, ciclos límite, soluciones armónicas y subarmónicas.

Con la finalización del total de ejmplos presentados en el artículo nos podemos percatar de lo interesante que es el temas, además de las ventajas con las que cuenta Phyton para resolver este tipo de ejercicios.

Podemos finalizar diciendo que Phyton y sus bibliotecas son buenas alternativas cuando se quiere resolver problemas que involucren soluciones numéricas, pues aunque se mencionaba que era difícil en el artículo llegar a soluciones, con ayuda de Phyton se lograron reproducir todas las gráficas, es una herramienta en quién confiar.

Bibliografía

- Integration and ODEs (scipy.integrate) — SciPy v1.0.0 Reference Guide. (2018). Docs.scipy.org. Recuperado el 21 de Marzo de 2018 desde https://docs.scipy.org/doc/scipy/reference/integrate.html
- JupyterLab Documentation — JupyterLab 1.0 Beta documentation. (2018). Jupyterlab.readthedocs.io. Recuperado el 21 de Marzo de 2018 desde http://jupyterlab.readthedocs.io/en/latest/
- Coupled spring-mass system — SciPy Cookbook documentation. (2018). Scipy-cookbook.readthedocs.io. Recuperado el 21 de Marzo de 2018 desde http://scipy-cookbook.readthedocs.io/items/CoupledSpringMassSystem.html

Apéndice

1. ¿Qué más te llama la atención de la actividad completa? ¿Que se te hizo menos interesante?
   Me gustó todo, pero fue bastante agradable que las gráficas me salieran tal cual el documuento, es una sensacion de satisfación. Podría decir no hubo nada que no me interesó.

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?
   La verdad no me lo imaginaba, pues estamos acostumbrados a verlos "bien comportados", pero el hecho de tratar este tipo de problemas con soluciones numéricas es muy interesante.

3. ¿Qué propondrías para mejorar esta actividad? ¿Te ha parecido interesante este reto?
   Fue muy interesante trabajar con estos temas, pero como mejora me gustaria más referencias para apoyarnos, sobre todo en los últimos dos temas.

4. ¿Quisieras estudiar mas este tipo de fenómenos no lineales?
   Estaría bien, ya que es muy interesante todo lo que se puede desglozar de ellos.