"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 6

"Sistema de Resortes Acoplados"

César Omar Ramírez Álvarez

Profr. Carlos Lizárraga Celaya

Hermosillo, Sonora Marzo 19 de 2018

Introdución

El presente informe es producto de la sexta práctica de la materia Física Computacional I, la cual tiene como objetivo aprender a modelar un fenómeno físico, en esta ocasión tomando de ejemplo un sistema de doble resorte acoplado, a través de una simulación.

Como es el caso de actividades anteriores, iniciaremos mostrando una breve síntesis sobre la teoría que hay detrás de nuestro sistema físico, basándonos en el artículo "Coupled Spring Equations" de los autores Fay y Graham. A diferencia de actividades anteriores que se trabajaba el código en Python mediante Jupyter Notebook, esta vez, exploramos a Jupyter Lab, un entorno que nos muestra un poco más de herramientas para Python. Presentaremos el código utilizado para resolver los diferentes problemas de una manera numérica con la ayuda de la biblioteca scipy.integrate.odeint, además mostraremos el error relativo existente entre lo calculado mediante el código y la solución analítica dada por el artículo. Visualizaremos una serie de gráficas generadas con los datos y finalizaremos con una conclusión general acerca de lo que fue la práctica mostrando reflexiones de lo aprendido en ella.

Síntesis

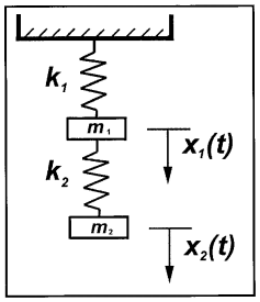
La perspectiva general con la que se toma a las ecuaciones diferenciales esta cambiando mucho, de realizar técnicas para encontrar soluciones analíticas a visualizar y resolver numéricamente éstas con las herramientas disponibles.

En este artículo, se investiga uno de los problemas mas interesantes de la Mecánica, y que ahora normalmente se utiliza para la introducción al estudio de ecuaciones diferenciales. Este problema es el de los dos resortes con dos masas conectadas en serie, colgando del techo. Si suponemos que las fuerzas restauradoras de los resortes obedecen la Ley de Hooke, estos dos grados de libertad nos dan un modelo de ecuaciones diferenciales lineales de segundo grado. Al sustituir una ecuación en la otra, el movimiento de las masas puede ser descrito por una ecuación diferencial lineal de cuarto grado.

Con estas ecuaciones, podemos investigar los movimientos de las dos masas, para saber si están sincronizadas o si son opuestas. Además, hacer esto permite tener una interpretación física clara de la fase, amplitud, periodicidad, y la sensibilidad del sistema a las condiciones iniciales cuando se introduce la no linealidad.

El Modelo de los Resortes Acoplados

El modelo consiste en dos resortes y dos masas. Un resorte con una constante k_1, está colgado del techo con una masa m_1 colgando de él. De aquí, se cuelga otro resorte con una constante k_2 y debajo de él colgando una masa m_2. Al tenerlos en reposo, los resortes se estiran una distancia, a la que nombramos x_1 y x_2.



Asumiendo la Ley de Hooke

Asumiendo que el sistema se mueve con oscilaciones pequeñas, podemos asumir que los resortes tendrán una fuerza restauradora dada por la Ley de Hooke, de la forma: $-k_1 l_1$ y $-k_2 l_2$ donde l_1 y l_2 son las elongaciones o comprensiones de los resortes. Como m_1 está atada a los dos resortes, en ella actúan las dos fuerzas restauradoras, mientras que m_2 solamente "siente" la fuerza restauradora del segundo resorte. Sin fricción, la Segunda Ley de Newton para estas dos masas es:

$$m_1\ddot{x}_1 = -k_1 x_1 - k_2(x_1 - x_2)$$
$$m_2\ddot{x}_2 = -k_2(x_2 - x_1)$$

Para encontrar una ecuación para x_1 que no involucre a x_2, resolvemos la ecuación de x_2, y sustituyendo esta ecuación en las ecuaciones anteriores se llega a la siguiente ecuación diferencial de cuarto grado:

$$m_1 m_2 x_1^{(4)} + (m_2 k_1 + k_2(m_1 + m_2))\ddot{x}_1 + k_1 k_2 x_1 = 0$$

Haciendo este mismo proceso, pero para x_2, se llega a la misma ecuación anterior. Es necesario solamente las posiciones y velocidades para poder determinar la solución.

Ejemplos con Masas Idénticas

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

Resolviendo el problema de una forma analítica, se llega a que las ecuaciones para las posiciones de las masas son:

$$x_1(t) = cos\sqrt{2}t$$
$$x_2(t) = 2cos\sqrt{2}t$$

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 primeros 4 ejemplos).

```
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
```

```
# Ejemplo 2.1

from scipy.integrate import odeint
import numpy as np

# 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('Ejercicio2.1.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],np.abs((w1[0]-(np.cos(np.sqrt(2.0)*t1)))/(np.cos(np.sqrt(2.0)*t1))),
               np.abs((w1[2]-(2.0*np.cos(np.sqrt(2.0)*t1)))/(2.0*np.cos(np.sqrt(2)*t1))), file=f)
```

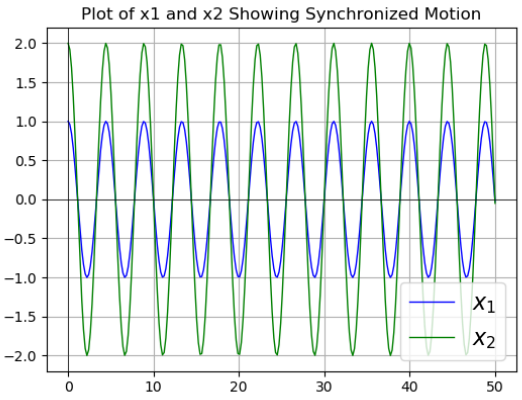
```
# Plot the solution that was generated

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, er1, er2 = loadtxt('Ejercicio2.1.dat', unpack=True)

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

grid(True)

lw = 1
```

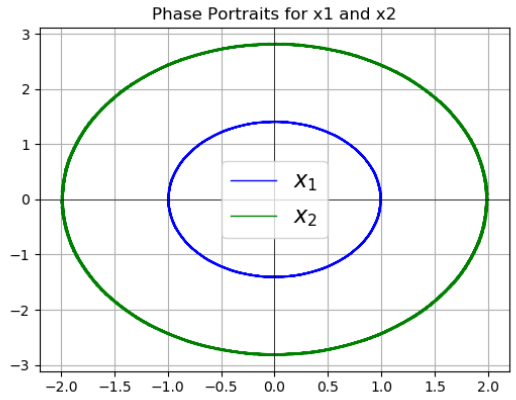
```
plot(t, x1, 'b', linewidth=lw)
plot(t, x2, 'g', linewidth=lw)

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

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Plot of x1 and x2 Showing Synchronized Motion')
savefig('G2.1a.png', dpi=100)
```

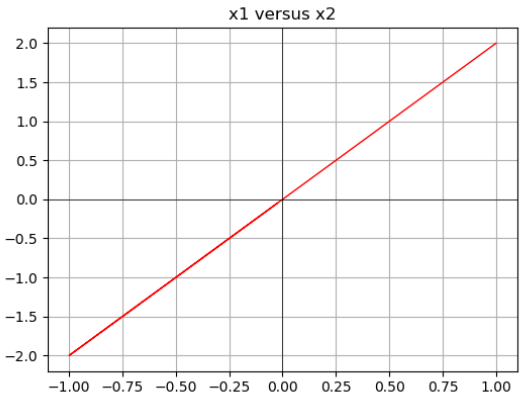
```
plot(x1, xy, 'b', linewidth=lw)
plot(x2, y2, 'g', linewidth=lw)

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

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Phase Portraits for x1 and x2')
savefig('G2.1b.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)

title('x1 versus x2')
savefig('G2.1c.png', dpi=100)
```

Las gráficas que se obtuvieron son:







Describiendo lo anterior, podríamos decir que tenemos un movimiento que esta sincronizado, ya que, aunque se tienen amplitudes diferentes, las masas se mueven en fase una con la otra lo cual es notorio en la primera gráfica. Las elipses de la segunda gráfica muestran

la variación de la amplitud y al graficar las posiciones una contra la otra nos genera una línea recta notable en la tercera gráfica.

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

La solución analítica es:

$$x_1(t) = -2cos2\sqrt{3}t$$
$$x_2(t) = cos2\sqrt{3}t$$

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

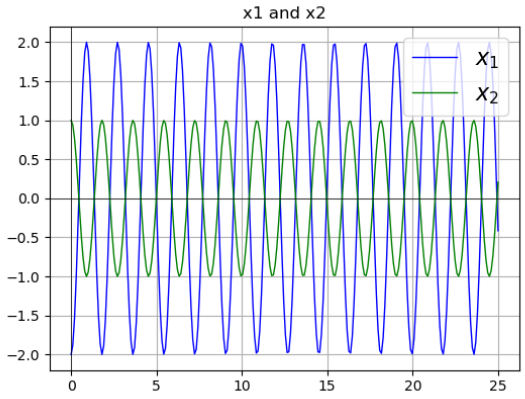
```
# Ejemplo 2.2

from scipy.integrate import odeint
import numpy as np

# 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 = 25.0
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('Ejercicio2.2.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],np.abs((w1[0]-(-2*(np.cos(2*np.sqrt(3.0)*t1))))/(-2*(np.cos(2*np.sqrt(3.0)*t1)))),
               np.abs((w1[2]-(np.cos(2*np.sqrt(3.0)*t1)))/(np.cos(2*np.sqrt(3.0)*t1))), file=f)
```

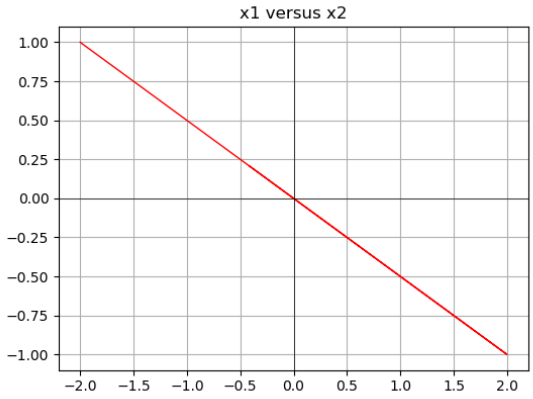
```
# Plot the solution that was generated

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, er1, er2 = loadtxt('Ejercicio2.2.dat', unpack=True)

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

grid(True)

lw = 1
```

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

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

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('x1 and x2')
savefig('G2.2a.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)

title('x1 versus x2')
savefig('G2.2b.png', dpi=100)
```

Las gráficas que se obtuvieron son:





En este caso, la primera masa se mueve hacia abajo mientras que la segunda se mueve hacia arriba, con el mismo periodo, pero en desfase.

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

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

```
# Ejemplo 2.3

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 = 1/2
y1 = 0.0
x2 = -1/2
y2 = 7/10

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 100.0
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]
w0 = [x1, y1, x2, y2]

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

with open('Ejercicio2.3.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('Ejercicio2.3.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)

title('Phase Plot for x1')
savefig('G2.3a.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)

title('Phase Plot for x2')
savefig('G2.3b.png', dpi=100)
```

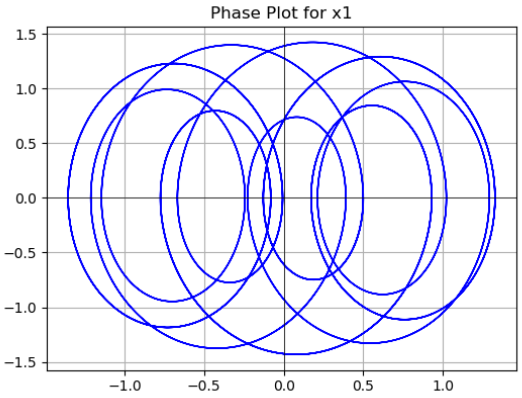
```
plot(t, x1, 'b', linewidth=lw)

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

y = [-2, -1,0,1,2]
plt.yticks(y)

title('Phase of x1')
savefig('G2.3c.png', dpi=100)
```

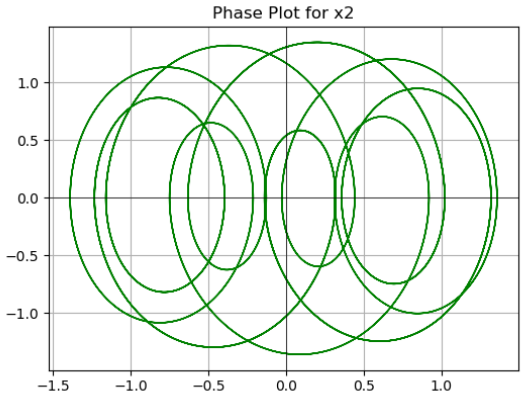
```
plot(t, x2, 'g', linewidth=lw)

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

y = [-2, -1,0,1,2]
plt.yticks(y)

title('Phase of x2')
savefig('G2.3d.png', dpi=100)
```

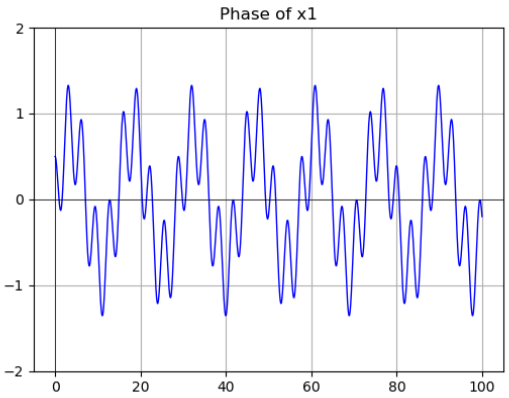
```
plot(t, x2, 'b', linewidth=lw)
plot(t, x1, 'g', linewidth=lw)

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

plt.xlim(0,50)
y = [-2, -1,0,1,2]
plt.yticks(y)

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Plot of x1 and x2')
savefig('G2.3e.png', dpi=100)
```

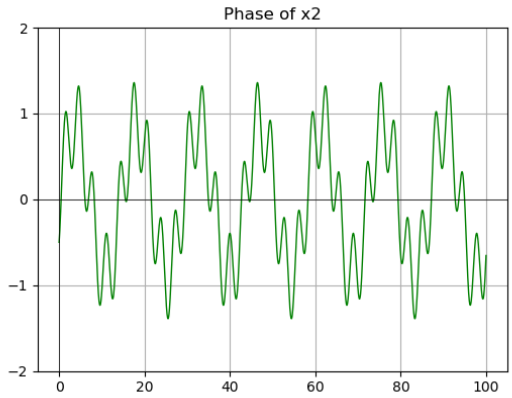
```
plot(x1, x2, 'r', linewidth=lw)

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

title('Plot of x1 versus x2')
savefig('G2.3f.png', dpi=100)
```

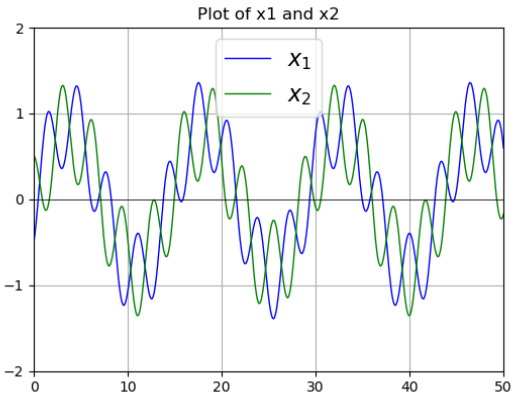
Las gráficas que se obtuvieron son:

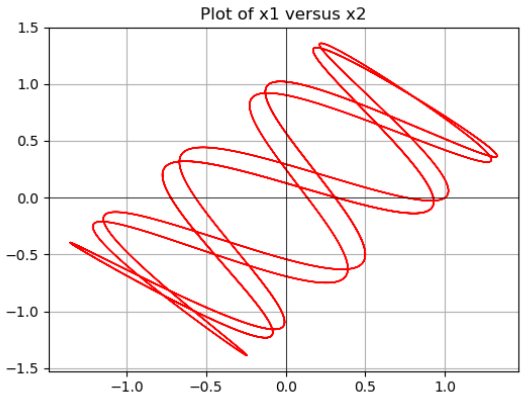












Es notorio en la observación de los resultados que las condiciones iniciales solo afectan a la amplitud y a la fase de las soluciones, mientras que las constantes de cada uno de los resortes influyen en la determinación de la frecuencia y el fenómeno del sistema.

Amortiguamiento

Los problemas más generales acerca de amortiguamiento es el de viscosidad, en donde la fuerza de amortiguamiento es proporcional a la velocidad. El amortiguamiento de la primera masa depende solamente de su velocidad y no en la de la segunda y viceversa. Asumimos que los coeficientes de amortiguamiento δ_1 y δ_2 son pequeños. El modelo se convierte y descrito por lo siguiente:

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

Para obtener la ecuación de movimiento para una de las variables de posición (x_1 o x_2) se realiza el mismo procedimiento que en la sección anterior, es decir, se sustituye en una ecuación en la ecuación anterior que le corresponde, así se obtiene una nueva ecuación diferencial de cuarto grado para ambas posiciones de x.

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

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

```
# Ejemplo 2.4

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 = 1/2
x2 = 2.0
y2 = 1/2

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50.0
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]
w0 = [x1, y1, x2, y2]

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

with open('Ejercicio2.4.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('Ejercicio2.4.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)

title('Phase Plot for x1')
savefig('G2.4a.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)

title('Phase Plot for x2')
savefig('G2.4b.png', dpi=100)
```

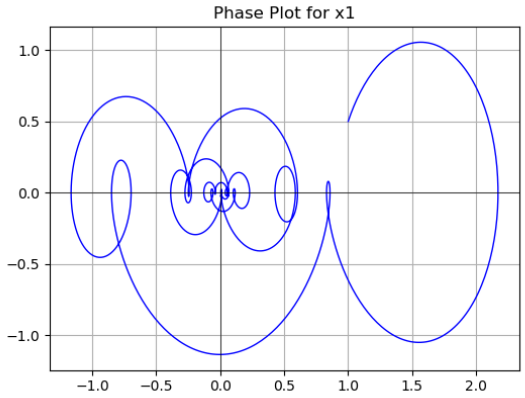
```
plot(t, x1, 'b', linewidth=lw)

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

title('Plot of x1')
savefig('G2.4c.png', dpi=100)
```

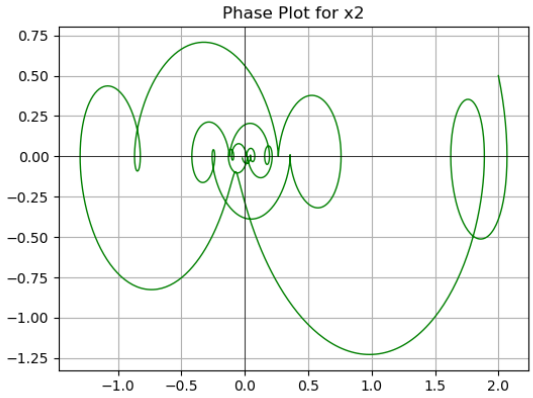
```
plot(t, x2, 'g', linewidth=lw)

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

title('Plot of x2')
savefig('G2.4d.png', dpi=100)
```

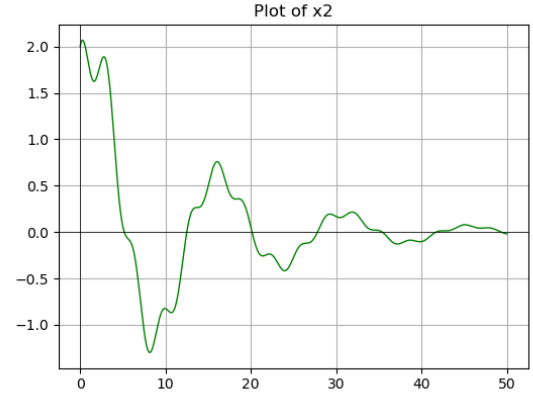
```
plot(t, x2, 'b', linewidth=lw)
plot(t, x1, 'g', linewidth=lw)

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

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Nearly Sychronized Motion')
savefig('G2.4e.png', dpi=100)
```

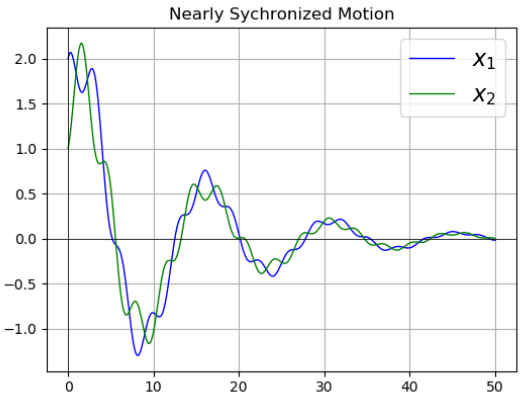
```
plot(x1, x2, 'r', linewidth=lw)

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

title('Plot of x1 versus x2')
savefig('G2.4f.png', dpi=100)
```

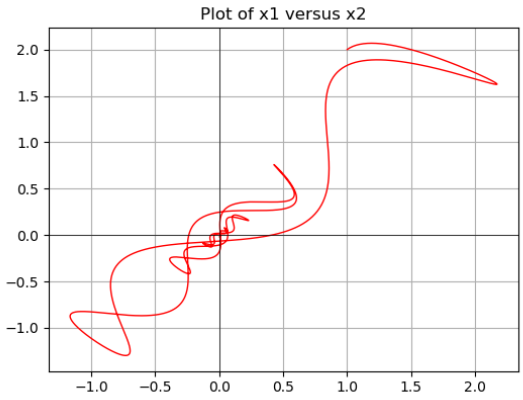
Las gráficas que se obtuvieron son:











De lo anterior es rescatable que, ahora existiendo amortiguamiento, las amplitudes disminuyen conforme el paso del tiempo. Se puede observar cierta sincronia al graficar x_1 y x_2, a pesar de que tienen distintas condiciones iniciales.

Solución Análitica y Solución Numérica

Contando con los dos tipos de soluciones: porporcionada por Python (numérica) y porporcionada por el artículo (analítica) es posible realizar el calculo del error relativo. Esto se hace mediante el valor real y nuestra aproximación obtenida. En los archivos creados en los ejercicios 2.1 y 2.2 se imprimió el error relativo, calculado de la siguiente manera: restando el valor real al aproximado dividiendo lo anterior entre el valor real.
El código para gráficar el error fue el siguiente:

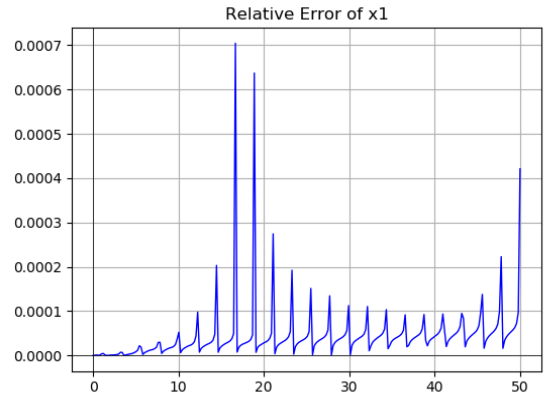
```
plot(t,er1, 'b', linewidth=lw)
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

title('Relative Error of x1')
savefig('G2.1d.png', dpi=100)
```

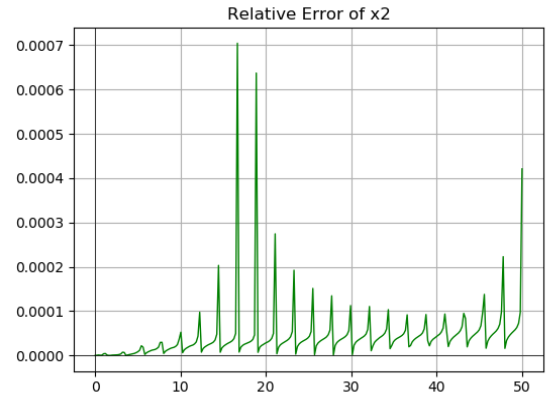
```
plot(t,er2, 'g', linewidth=lw)
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

title('Relative Error of x2')
savefig('G2.1e.png', dpi=100)
```

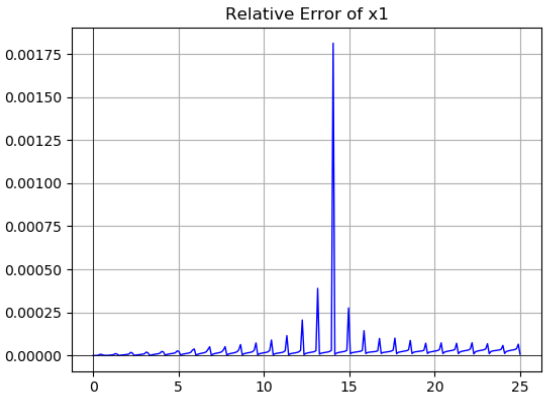
```
plot(t,er1, 'b', linewidth=lw)
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

title('Relative Error of x1')
savefig('G2.2c.png', dpi=100)
```

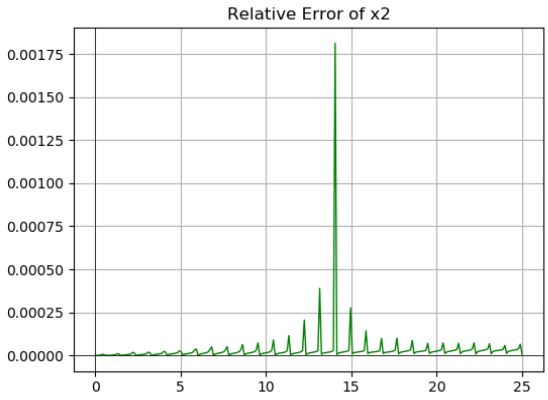
```
plot(t,er2, 'g', linewidth=lw)

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

title('Relative Error of x2')
savefig('G2.1e.png', dpi=100)
```

Mientras que las gráficas son:









Podemos decir que para ambos ejemplos los errores son muy pequeños y muy idénticos entre x_1 y x_2.

Conclusión

Jupyter Lab como entorno de trabajo, me parecio una mas eficaz que cuando haciamos uso de Jupyter Notebook, pues en este nuevo entorno se pueden realizar más cosas y tener un control de lo que estamos generando con el código. Además con el uso de la biblioteca para integrar fue posible resolver de manera muy rápida las ecuaciones diferenciales propuestas en los ejercicios.

El tema fue bastante interesante, creo que fue muy buena ejemplificación para el uso de las nuevas herramientas y conocimientos alcanzados, ya que, es un tema común en la Licenciatura que se aborda como estudio en materias como Mecánica y es bueno tener un repaso y conocimiento del mismo tanto de manera teórica como analítica. Sin duda Python es muy habil para hacer lectura de archivos y realizar calculos numéricos con los mismos.

Bibliografía

- Integration and ODEs (scipy.integrate) — SciPy v1.0.0 Reference Guide. (2018). Docs.scipy.org. Recuperado el 12 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 12 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 12 de Marzo de 2018 desde http://scipy-cookbook.readthedocs.io/items/CoupledSpringMassSystem.html

Apéndice

1. ¿En general te pareció interesante esta actividad de modelación matemática? ¿Qué te gustó mas? ¿Qué no te gustó?
   Me pareció bastante interesante el tema y las gráficas que se generaban con los datos, pero siento que le faltaron un poco más de referencia y explicación a lo que se iba a realizar.

2. La cantidad de material te pareció ¿bien?, ¿suficiente?, ¿demasiado?
   Fue lo suficiente, aunque parecia mucha en un principio.

3. ¿Cuál es tu primera impresión de Jupyter Lab?
   A diferencia de Notebook, me parecio más fácial de entender y con un mejor diseño, además cuenta con mejores herramientas.

4. Respecto al uso de funciones de SciPy, ¿ya habías visto integración numérica en tus cursos anteriores? ¿Cuál es tu experiencia?.
   Si eh visto, en Programación y Análisis Numérico pero en FORTRAN con Trapecios y Simpson.

5. El tema de sistema de masas acopladas con resortes, ¿ya lo habías resuelto en tu curso de Mecánica 2?
   Si, pero sin tantas condiciones fue explicación sencilla.

6. ¿Qué le quitarías o agregarías a esta actividad para hacerla más interesante y divertida?
   Sugiero mas referencias y explicación de lo que vamos a realizar, a veces me pierdo.