Universidad De Sonora

División de Ciencias Exactas y Naturales

Licenciatura en Física

Física Computacional I

Actividad 6

"Sistema de resortes acoplados"

Michelle Contreras Cossio

Profr. Carlos Lizárraga Celaya

Hermosillo, Sonora 19 de Marzo de 2018

1 Introducción

El presente reporte muestra el procedimiento y resultados obtenidos al realizar la actividad #6 de la clase Física Computacional I. El objetivo de la práctica es continuar con el uso de phyton, enfocandonos en la simulación de cierto fenómeno físico, para poder comparar soluciones numéricas con analíticas.

Durante esta semana se trabajó con jupyter lab, otra forma de trabajar con phyton, donde tenemos un mayor acercamiento a los archivos con los que trabajamos, una interfaz un poco diferente y con otras funcionalidades.

Además, se utilizó el artículo de "Coupled spring equations" para simular, con ayuda de phyton, sistemas de resortes acoplados, para obtener gráficas identicas a las del artículo.

2 Síntesis de "Coupled spring equations"

En esta sección mostramos una síntesis de artículo "Coupled spring equations" por Temple H. Fay y Sarah Duncan Graham. Además se añade el código utilizado para resolver estos mismos problemas, haciendo uso de jupyter lab.

2.1 Introducción

En este artículo, se trata el problema de dos resortes y dos masas, conectados en serie y suspendidos desde el techo. Para resolver este fenómeno, se hace uso de la Ley de Hooke, para la fuerza restauradora del resorte, lo que resulta en la solución de ecuaciones diferenciales lineales de segundo orden, sin embargo, se pueden llegar a convertir en ecuaciones diferenciales lineales de hasta cuarto orden.

A partir de estas ecuaciones se pueden tener muchas interpretaciones físicas, sobre la fase o desfase, además que se pueden agregar términos como uno no lineal, que pretende mostrar algo físicamente más viable.

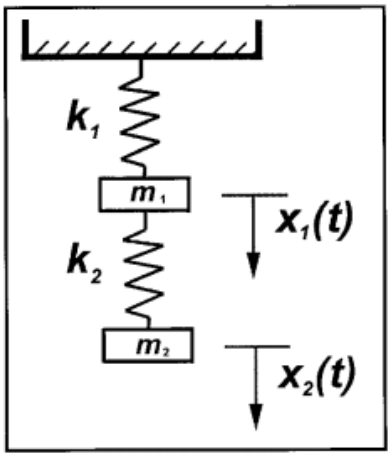
En resumen, podemos modificar fácilmente variables como la periodicidad, fase, amplitud, entre otras, modificando simplemente las ecuaciones.

2.2 El modelo de resortes acoplados

Este modelo consiste en dos resortes y dos masas, valores de k y m diferentes, colgados al techo y movidos desde su posición de equilibrio en x.

2.2.1 Asumiendo la Ley de Hooke

Tomando como cierta la Ley de Hooke, cada resorte estirados tendrá una fuerza restauradora de -kx, donde k es la constante del resorte y x es lo que se estiró.



La masa superior siente dos fuerzas, puesto que se encuentra entre dos resortes, la propia al resorte superior y una fuerza hacia arriba por la resistencia del segundo resorte para ser estirado. Mientras que la masa inferior únicamente siente la fuerza del resorte al cual se encuentra conectado. De esta manera, gracias a la Ley de Newton, podemos representar lo dicho en dos ecuaciones:

$$m_1\ddot{x}_1 = -k_1x_1 - k_2(x_1 - x_2) \tag{1}$$

$$m_2\ddot{x}_2 = -k_2(x_2 - x_1) \tag{2}$$

Si se buscara encontrar una ecuación para x_1, de modo que no se involucrara a x_2, y viceversa se obtienen dos ecuaciones lineales de cuarto grado, a las cuales se llega, despejando una variable de la primera ecuación y sustituyendo en la segunda:

$$m_1m_2x_1^{(4)} + (m_2k_1 + k_2(m_1 + m_2))\ddot{x}_1 + k_1k_2x_1 = 0$$
$$m_1m_2x_2^{(4)} + (m_2k_1 + k_2(m_1 + m_2))\ddot{x}_2 + k_1k_2x_2 = 0$$

.

Ecuaciones con las cuales, solamente se hace uso de los desplazamientos y velocidades

iniciales. Así pues, una alternativa que se tiene es convertir el sistema de dos ecuaciones de segundo orden a uno de 4 ecuaciones de cuarto orden. Sin embargo, en los ejemplos se trabajara con el primer sistema.

2.2.2 Ejemplos

Los siguientes ejemplos muestran el uso de dos masas $m_1 = m_2 = 1$. Sin fricción ni fuerza externa, haciendo uso de las ecuaciones (1) y (2).

- **Ejemplo 2.1**

  Describir el movimiento de los resortes con k_1=6, k_2=4 y condiciones iniciales $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (1, 0, 2, 0)$.

  Resolviendo las ecuaciones, se llegó a que la solución analítica para este problema es:

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

  Este movimiento es sincronizado y se encuentra en fase, las masas tienen el mismo período, esto se puede ver en la gráficas creadas, mostradas a continuación.

  – **Solución numérica con Phyton:**

    NOTA: El código mostrado a continuación, así como el que produce las gráficas, se utilizó de manera idéntica para los ejemplos 2.1, 2.2 y 2.3, cambiando únicamente las condiciones iniciales (celda 2):

    Lo que hace este código es crear vectores que obtengan los valores para las velocidades, posiciones, masas, constantes de los resortes, etc.

```
In [1]: 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
```

Posteriormente se tiene una sección que permite dar los valores iniciales de masas, constantes, etc. Se crea un archivo de datos, con tiempos que van cambiando según se asigne, que contiene tiempos, velocidad y posición a cierto tiempo y el error (en este caso y en el ejemplo 2.2).

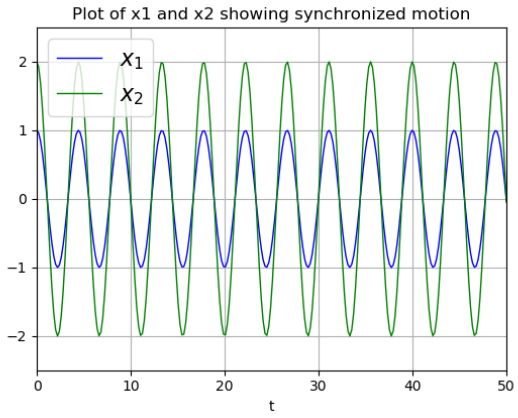
In [2]:
```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
import numpy as np

# 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('ej2_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)*t1)))/(np.cos(np.sqrt(2)*t1))),
               np.abs((w1[2]-(2*np.cos(np.sqrt(2)*t1)))/(2*np.cos(np.sqrt(2)*t1))),
```

A partir de ese archivo se crearon las gráficas, utilizando algunas de las variables que contenga.

Así se produjo el movimiento de x_1 y de x_2 a través del tiempo:

```
# Plot the solution that was generated

import matplotlib.pyplot as plt
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('ej2_1.dat', unpack=True)

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

xlabel('t')
grid(True)
#hold(True)
lw = 1
plt.xlim(0,50)
plt.ylim (-2.5,2.5)

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

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Plot of x1 and x2 showing synchronized motion')
savefig('ej2_1.1.png', dpi=100)
```

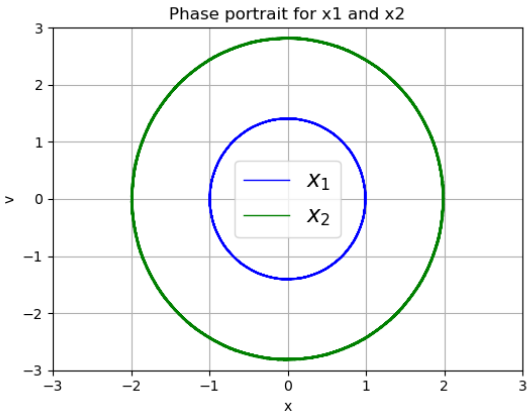
Dando como resultado:



Así se produjo la fase de x_1 y de x_2 (v vs x):

```
# Plot the solution that was generated

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

t, x1, y1, x2, y2, e1,e2 = loadtxt('ej2_1.dat', unpack=True)

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

xlabel('x')
ylabel('v')
grid(True)
#hold(True)
lw = 1
plt.xlim(-3,3)
plt.ylim (-3,3)

plot(x1, y1,'b', linewidth=lw)
plot(x2, y2, 'g', linewidth=lw)

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Phase portrait for x1 and x2')
savefig('ej2_1.2.png', dpi=100)
```

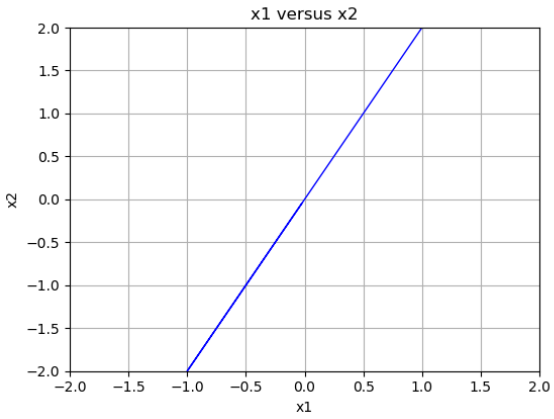
Dando como resultado:



Así se produjo x_1 y vs x_2:

```
# Plot the solution that was generated

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

t, x1, xy, x2, y2, e1,e2 = loadtxt('ej2_1.dat', unpack=True)

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

xlabel('x1')
ylabel('x2')
grid(True)
#hold(True)
lw = 1
plt.xlim(-2,2)
plt.ylim (-2,2)

plot(x1,x2, 'b', linewidth=lw)

title('x1 versus x2')
savefig('ej2_1.3.png', dpi=100)
```

Dando como resultado:



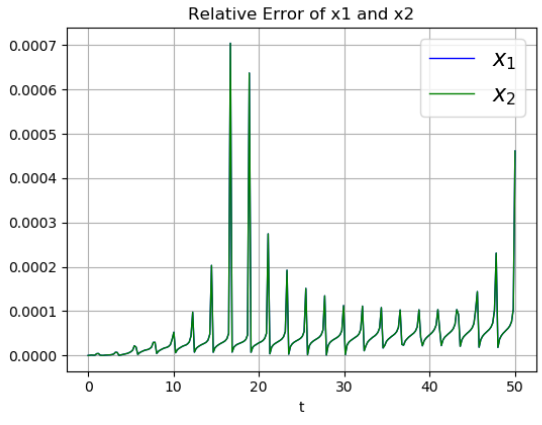
– **Calculo del error**

Para calcular el error, primeramente, se agregaron al archivo creado dos columnas,una para el error con respecto a x_1 y otra para el error con respecto a x_2, esos errores se calcularon restando a la x correspondiente a cierto tiempo (valor numérico), la función solución evaluada en el mismo tiempo (valor analítico),y dividiendolo entre este valor analítico.

```
with open('ej2_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)*t1)))/(np.cos(np.sqrt(2)*t1))),
               np.abs((w1[2]-(2*np.cos(np.sqrt(2)*t1)))/(2*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
%matplotlib inline

t, x1, y1, x2, y2, e1, e2 = loadtxt('ej2_1.dat', unpack=True)

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

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

plot(t, e1, 'b', linewidth=lw)

plot(t, e2, 'g', linewidth=lw)

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Relative Error of x1 and x2')
savefig('error2.1.png', dpi=100)
```

Posteriormente, se graficaron ambos errores con respecto del tiempo:



- **Ejemplo 2.2**

Describir el movimiento de los resortes con k_1=6, k_2=4 y condiciones iniciales $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0))$ = (-2, 0, 1, 0).

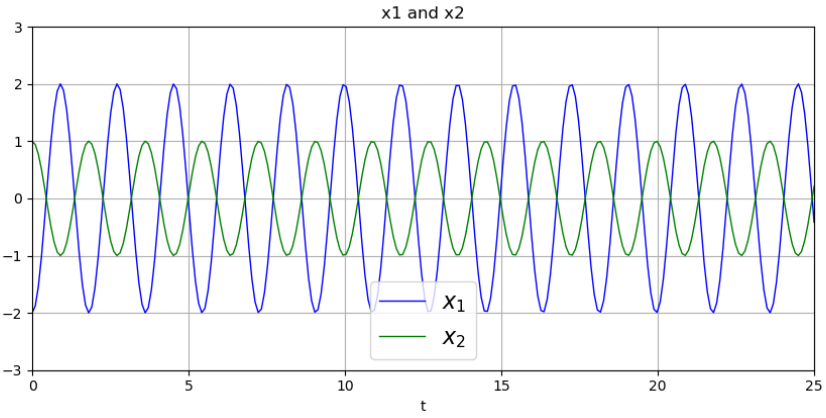
Resolviendo las ecuaciones, se llegó a que la solución analítica para este problema es:

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

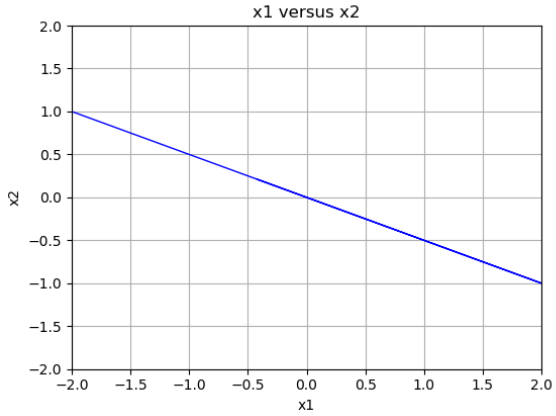
Lo que representa que cuando una masa se mueve para arriba, la otra se mueve para abajo, pero tienen el mismo período, con un desfase de 180°

- **Solución numérica con Phyton**

  Se utilizó el mismo código utilizado en el ejemplo anterior, pero únicamente se produjeron las gráficas de el movimiento de x_1 y de x_2 a través del tiempo:

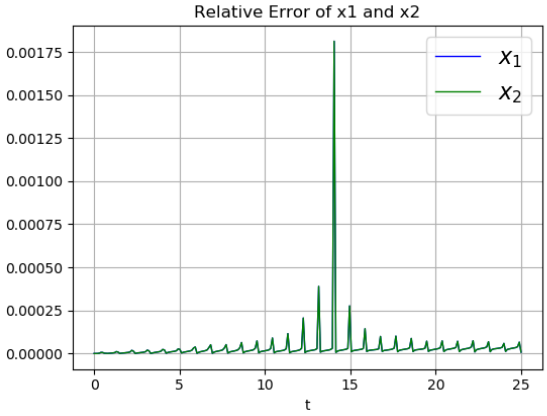


  y de x_1 y vs x_2:



- **Cálculo del error**

El error se produjo de la misma manera que en el ejemplo anterior, utilizando ahora las nuevas soluciones analíticas, la gráfica obtenida fue la siguiente:



- **Ejemplo 2.3**

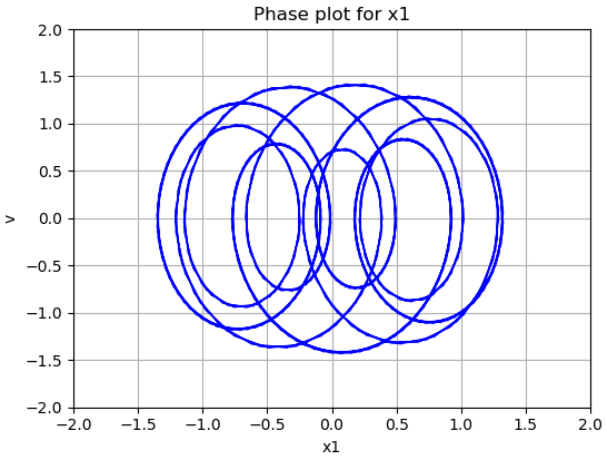
  Describir el movimiento de los resortes con k_1=0.4, k_2=1.808 y condiciones iniciales $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (1/2,\ 0,\ \text{-}1/2,\ 7/10)$.

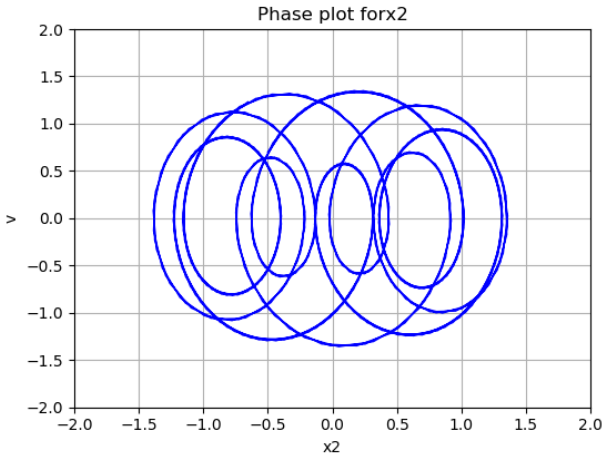
  Cambiando estos valores iniciales, únicamente va a cambiar el período y la frecuencia, así como la amplitud y fase en la soluciones. Con los programas que creamos podemos verificar eso al variar los valores de k.

  – **Solución numérica con Phyton**

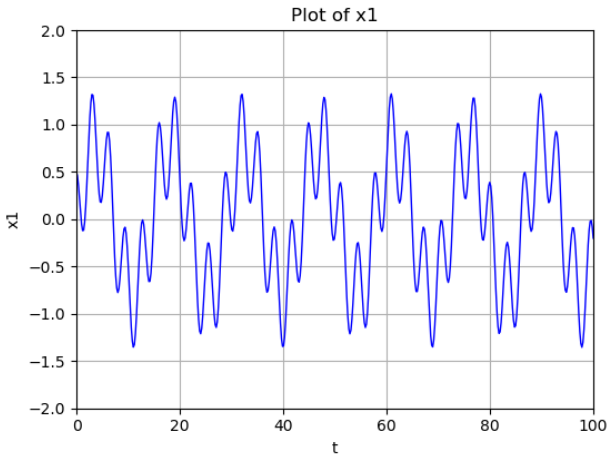
    Se realizaron básicamente las mismas gráficas que se crearon en los ejemplos anteriores.

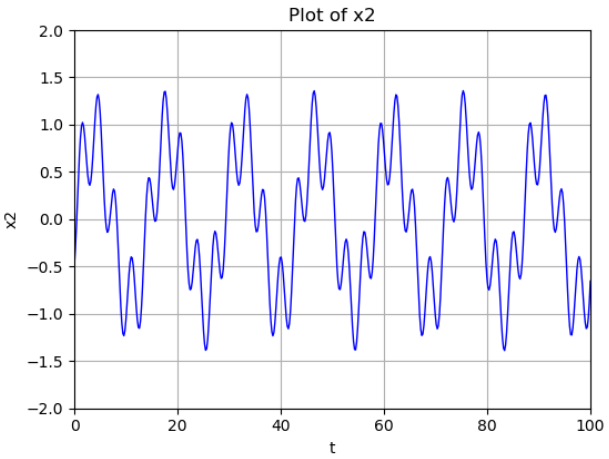
    Primeramente se crearon las gráficas de fase, una para x_1 y otra para x_2, esta vez se encuentra cada uno en una gráfica.



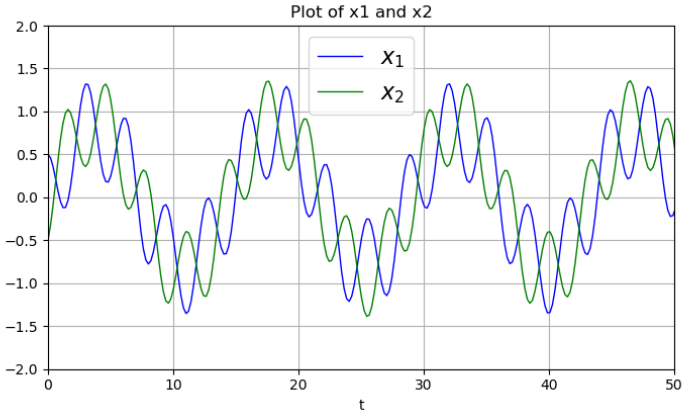


Posteriormente el plot de x_1 y x_2, por separado con respecto al tiempo.

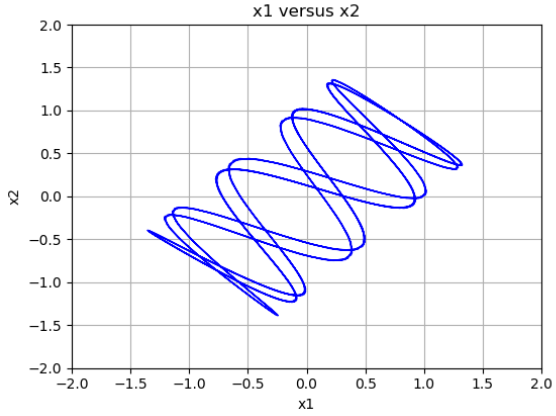




El movimiento de x_1 y x_2 con respecto al tiempo, en una sola.



Plot de x_1 vs x_2.



2.2.3 Amortiguamiento

El tipo de amortiguamiento (o fricción para el caso en el que las masas se encuentran sobre alguna superficie) con el que estaremos trabajando es el viscoso, que depende únicamente de la velocidad. En este caso, que tenemos dos resortes, el amortiguamiento de la primera masa depende únicamente de su velocidad y no de la velocidad de la otra masa y viceversa.

Así, añadiendo el término de amortiguamiento, las ecuaciones quedan:

$$m_1\ddot{x}_1 = -\delta_1\dot{x}_1 - k_1x_1 - k_2(x_1 - x_2) \tag{3}$$

$$m_2\ddot{x}_2 = -\delta_2\dot{x}_2 - k_2(x_2 - x_1) \tag{4}$$

De igual manera que en el movimiento de dos resortes acoplados simples, se puede modificar el sistema que tenemos de dos ecuaciones de segundo grado a un sistema de cuatro ecuaciones de cuarto grado, sin embargo, trabajaremos con el que ya tenemos.

- **Ejemplo 2.4**

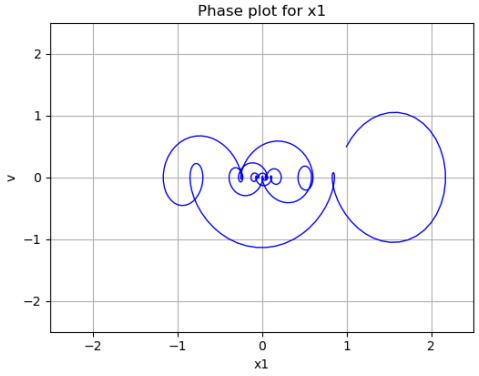
  Describir el movimiento de los resortes con $m_1 = m_2 = 1$, k_1=0.4, k_2=1.808, constantes de amortiguamiento δ_1= 0.1, δ_2= 0.2 y condiciones iniciales $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0))$ = (1, 1/2, 2, 1/2).

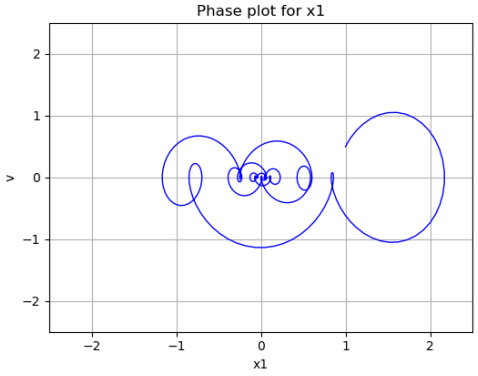
  El amortiguamiento va a causar que poco a poco la amplitud del movimiento disminuya.

  - **Solución numérica con Phyton**

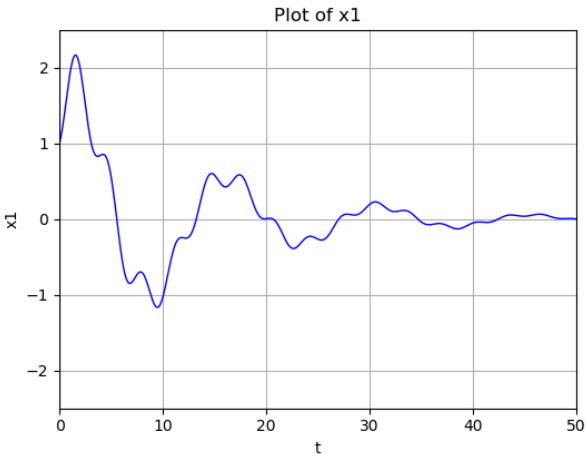
    El código utilizado sigue siendo el mismo, únicamente se tiene que en este caso si existirán coeficientes de fricción/ amortiguamiento. Se crearon las mismas gráficas del ejemplo anterior.

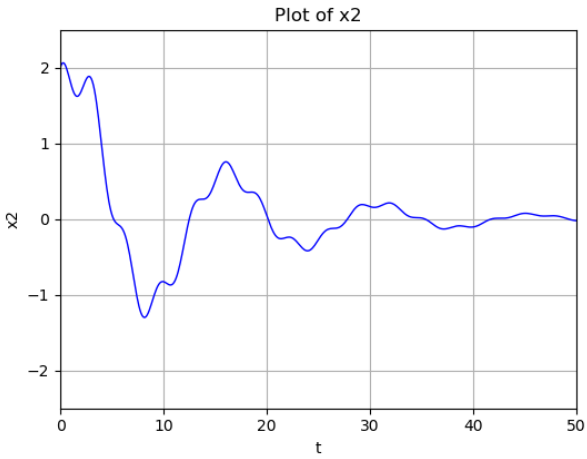
    Gráficas de fase, una para x_1 y otra para x_2.



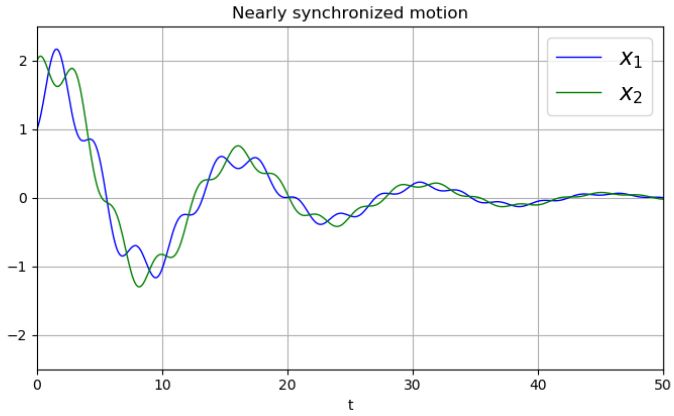


Plot de x_1 y x_2

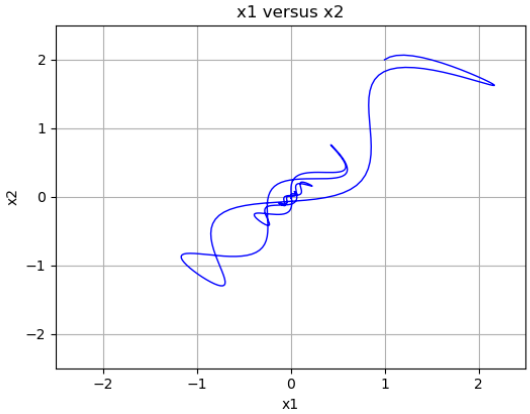




El movimiento de x_1 y x_2 con respecto al tiempo



Plot de x_1 vs x_2.



3 Conclusiones

Primeramente, me gustaría hablar del entorno Jupyter Lab, me pareció muy bueno, ya que cuenta con todo lo que se necesita en una sola ventana, cuenta con un navegador de archivos que es realmente útil. Me gustó más que el anterior.

Por otro lado, fue interesante trabajar por primera vez con simulación numérica, cosa que no se había trabajado como tal, pudiendo relacionar cosas que ya hemos aprendido, tanto en esta como en otras materias.

4 Bibliografía

- Ley de elasticidad de Hooke (2018). Consultado: 25 de Marzo del 2018, de Wikipedia. Sitio web: https://es.wikipedia.org/wiki/Ley_de_elasticidad_de_Hooke
- Couple spring equations (2003).Temple H. Fay, Sarah Duncan Graham. Consultado: 27 de Marzo del 2018, de Oregon State University. Sitio web: http://math.oregonstate.edu/g̃ibsonn/Teaching/MTH323-010S15/Supplements/coupled_spring.pdf

5 Apéndice

1. ¿En general te pareció interesante esta actividad de modelación matemática? ¿Qué te gustó mas? ¿Qué no te gustó?

   Si me pareció interesante, me gustó poder resolver problemas de una manera numérica, como en la clase de análisis numérico, ya que en ocasiones dejamos de lado estas soluciones, que en realidad son muy útiles.

2. La cantidad de material te pareció ¿bien?, ¿suficiente?, ¿demasiado?

   Me pareció suficiente, aunque tarde bastante en realizar la actividad.

3. ¿Cuál es tu primera impresión de Jupyter Lab?

   Bastante útil, manejable y todo se encuentra en una sola ventana, cosa que me agrada mucho, tener todo a la mano.

4. Respecto al uso de funciones de SciPy, ¿ya habías visto integración numérica en tus cursos anteriores? ¿Cuál es tu experiencia?.

Si, en cálculo y análisis numérico. Este método te lo realiza todo en automático, ya que antes lo teníamos que realizar todo, sin una función, ya fuera a mano o computadora.

5. El tema de sistema de masas acopladas con resortes, ¿ya lo habías resuelto en tu curso de Mecánica 2?

   Si, pero creo que con amortiguamiento no.

6. ¿Qué le quitarías o agregarías a esta actividad para hacerla más interesante y divertida?

   Me pareció justo el tema y el contenido que tiene.