Universidad De Sonora

División de Ciencias Exactas y Naturales

Licenciatura en Física

Física Computacional I

Actividad 7

"Sistema de resortes acoplados Final"

Michelle Contreras Cossio

Profr. Carlos Lizárraga Celaya

Hermosillo, Sonora 24 de Marzo de 2018

1 Introducción

El presente reporte muestra el procedimiento y resultados obtenidos al realizar la actividad #7 de la clase Física Computacional I. Durante esta práctica se retoma el artículo utilizado en la práctica anterior, continuando con los siguientes apartados, por lo que el objetivo prevalece el cual es continuar con el uso de Python, enfocándonos en la simulación de cierto fenómeno físico, para poder comparar soluciones numéricas con analíticas.

De igual manera, continuaremos trabajando con jupyter lab utilizando el archivo de "Coupled spring equations", trabajando con el sistema de resortes acoplados ya sea sencillo, con amortiguamiento, con algún coeficiente no lineal y con fuerzas externas.

A continuación se presenta la síntesis completa 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.

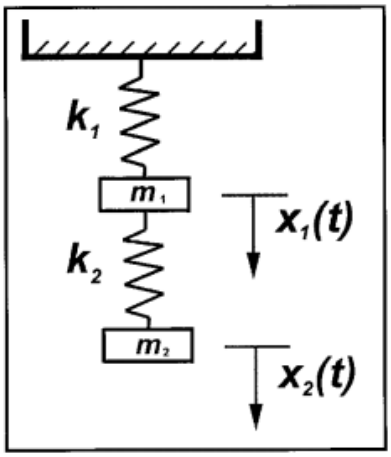
En resumen, podemos modificar fácilmente variables como la periodicidad, fase, amplitud, entre otras, modificando simplemente las ecuaciones.

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 Python:**

    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).

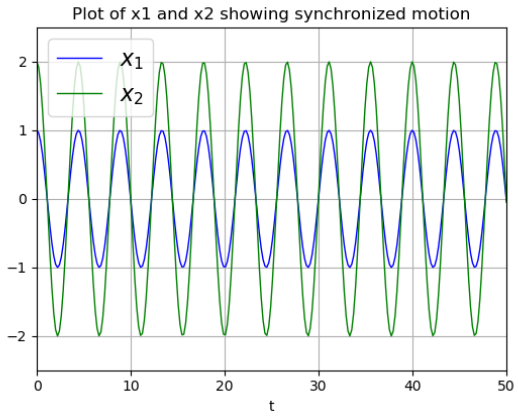
In [2]:
```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
import numpy as np

# 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)
```

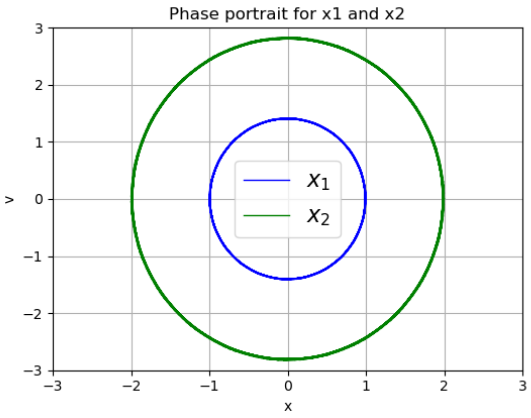
Dando como resultado:



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)
```

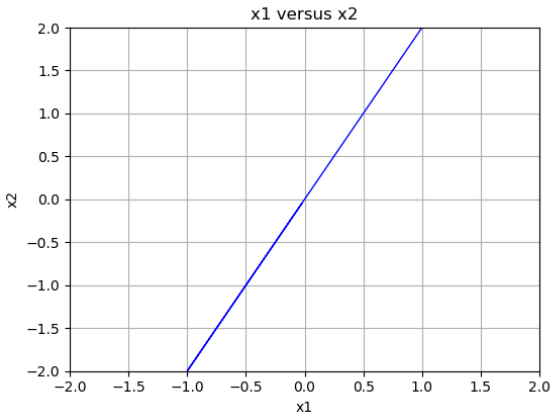
Dando como resultado:



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:



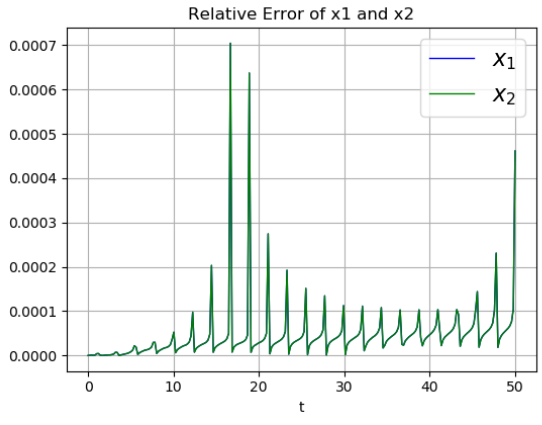
– **Calculo del error**

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).

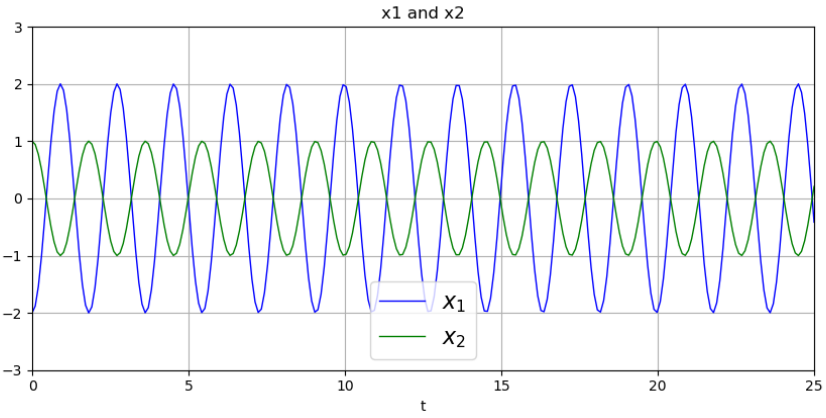
Resolviendo las ecuaciones, se llegó a que la solución analítica para este problema es:

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

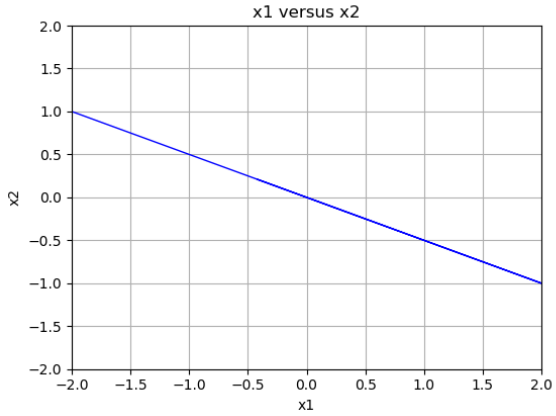
Lo que representa que cuando una masa se mueve para arriba, la otra se mueve para abajo, pero tienen el mismo período, con un desfase de 180°

- **Solución numérica con Python**

  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:

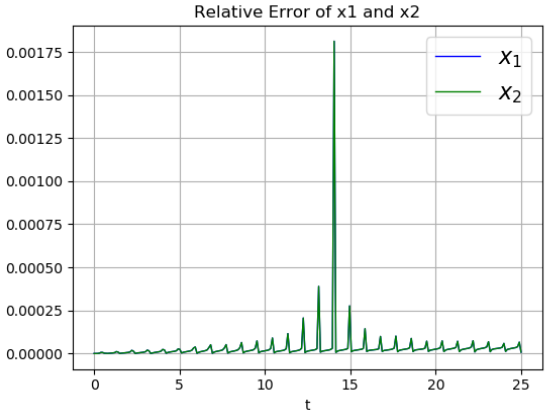


  y de x_1 y vs x_2:



- **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**

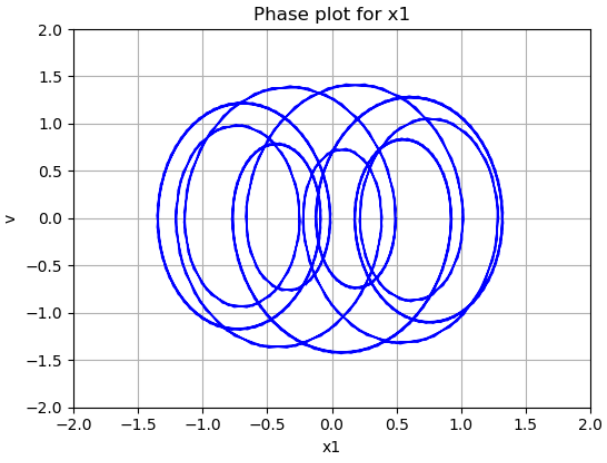
  Describir el movimiento de los resortes con k_1=0.4, k_2=1.808 y condiciones iniciales $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (1/2,\ 0,\ \text{-}1/2,\ 7/10)$.

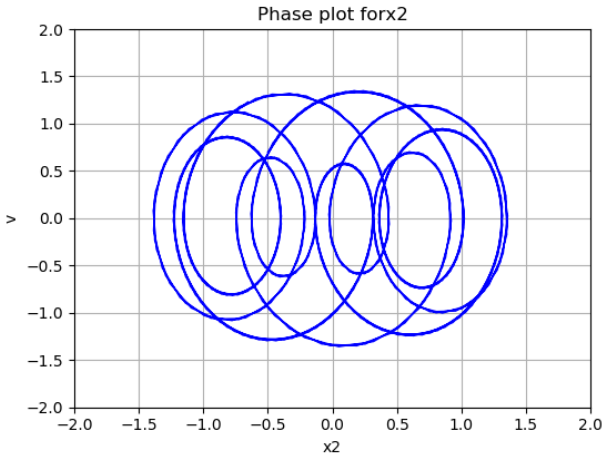
  Cambiando estos valores iniciales, únicamente va a cambiar el período y la frecuencia, así como la amplitud y fase en la soluciones. Con los programas que creamos podemos verificar eso al variar los valores de k.

  – **Solución numérica con Python**

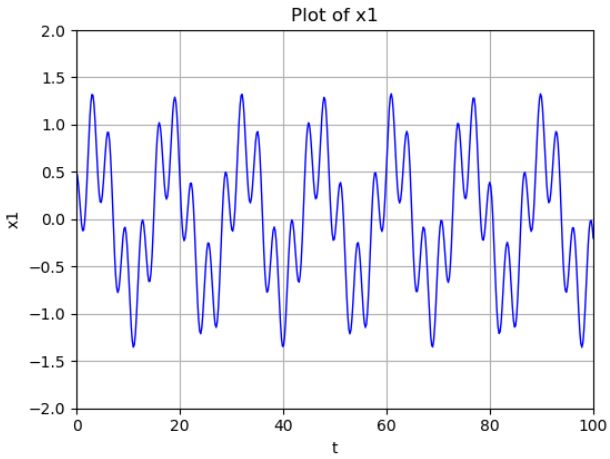
    Se realizaron básicamente las mismas gráficas que se crearon en los ejemplos anteriores.

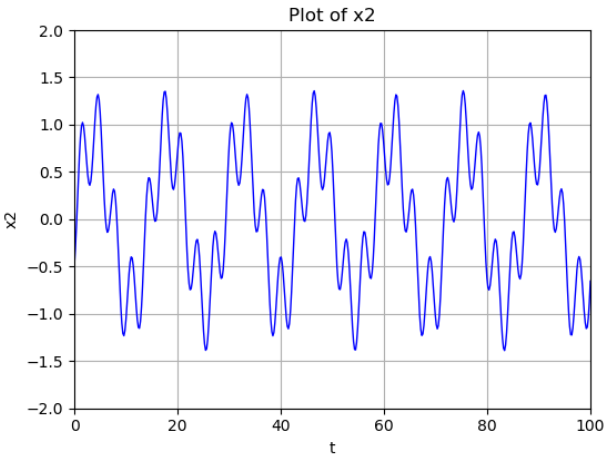
    Primeramente se crearon las gráficas de fase, una para x_1 y otra para x_2, esta vez se encuentra cada uno en una gráfica.



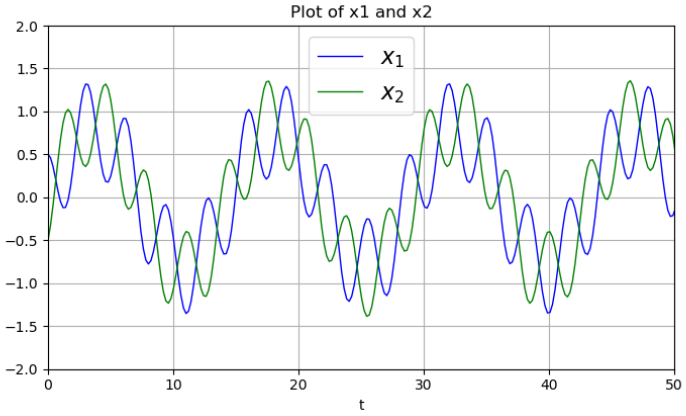


Posteriormente el plot de x_1 y x_2, por separado con respecto al tiempo.

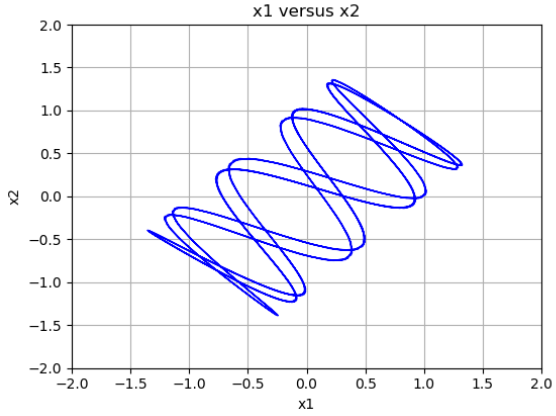




El movimiento de x_1 y x_2 con respecto al tiempo, en una sola.



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**

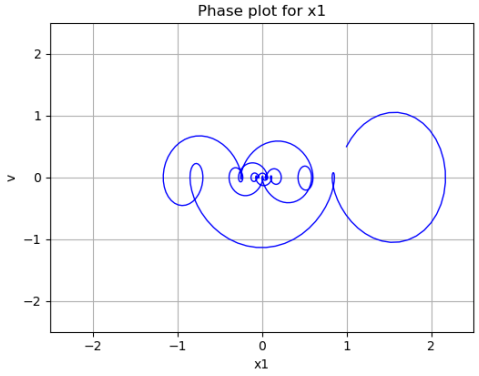
  Describir el movimiento de los resortes con $m_1 = m_2 = 1$, k_1=0.4, k_2=1.808, constantes de amortiguamiento δ_1= 0.1, δ_2= 0.2 y condiciones iniciales $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0))$ = (1, 1/2, 2, 1/2).

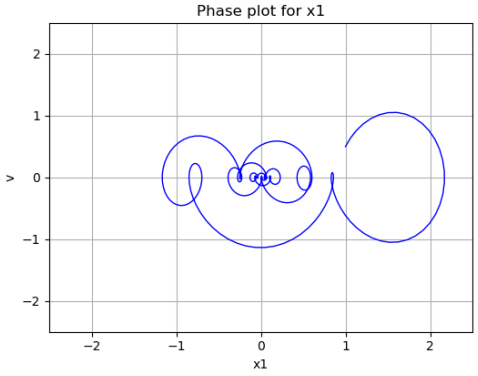
  El amortiguamiento va a causar que poco a poco la amplitud del movimiento disminuya.

  - **Solución numérica con Python**

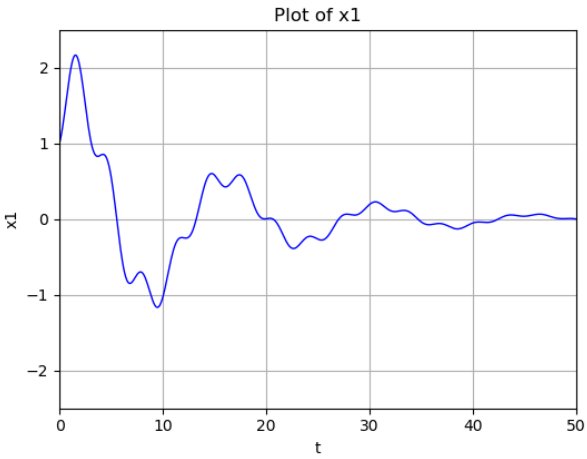
    El código utilizado sigue siendo el mismo, únicamente se tiene que en este caso si existirán coeficientes de fricción/ amortiguamiento. Se crearon las mismas gráficas del ejemplo anterior.

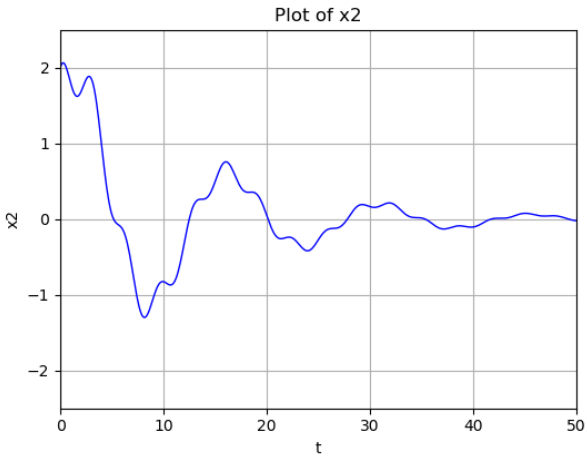
    Gráficas de fase, una para x_1 y otra para x_2.



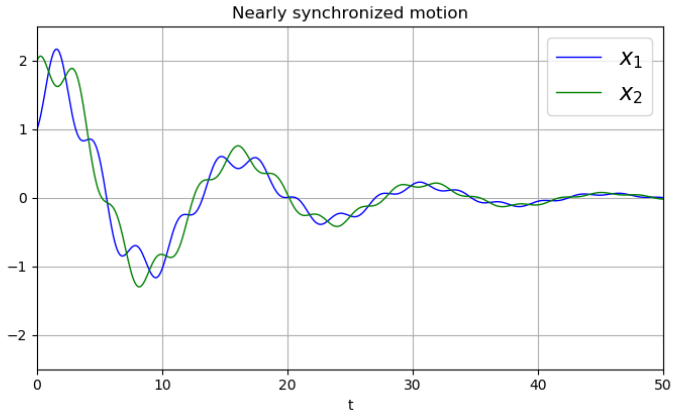


Plot de x_1 y x_2

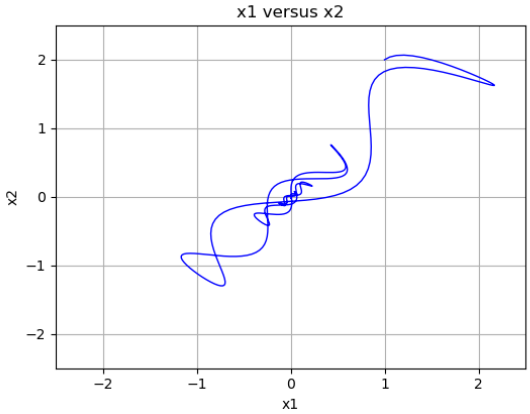




El movimiento de x_1 y x_2 con respecto al tiempo



Plot de x_1 vs x_2.



2.3 Agregando no linealidad

Si asumimos que las fuerzas de restauración son no lineales, que es muy probable si nos aproximamos a un sistema real, se modifican las ecuaciones que teniamos, de la siguiente manera:

$$m_1\ddot{x}_1 = -\delta_1\dot{x}_1 - k_1x_1 + \mu_1x_1^3 - k_2(x_1 - x_2) + \mu_2(x_1 - x_2)^3 \tag{5}$$

$$m_2\ddot{x}_2 = -\delta_2\dot{x}_2 - k_2(x_2 - x_1) + \mu_2(x_2 - x_1)^3 \tag{6}$$

Sin embargo, este modelo es mucho más complicado que uno lineal, tanto que si tomamos intervalos de tiempo grandes, las soluciones numéricas dejan de acercarse a la solución real.

- **Ejemplo 3.1**

  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, δ_2= 0, coeficientes no lineales μ_1=-1/6, μ_2=-1/10 y condiciones iniciales $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0))$ = (1, 0, -1/2, 0).

  El movimiento es oscilatorio y parece periódico, no hay amortiguamiento que disminuya la amplitud, aparentemente tienen un desfase de 180°, debido que es no lineal, el modelo tiene sensibilidad a las condiciones iniciales.

  - **Solución numérica con Python**

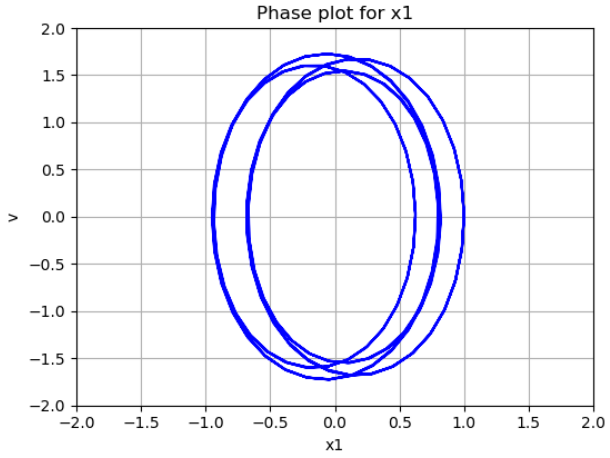
    El código utilizado fue básicamente el mismo que el mostrado en el ejemplo 2.1, a continuación se muestran las modificaciones a tal código (utilizado también en el ejemplo 3.2 y 3.3:

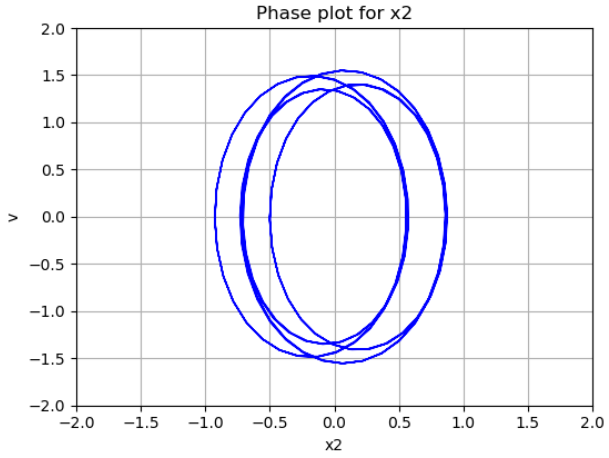
In [1]:
```
def vectorfield(w, t, p):
    """
    Defines the differential equations for the coupled spring-mass system.

    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,u1,u2 = p

    # Create f = (x1',y1',x2',y2'):
    f = [y1,
         (-b1 * y1 - k1 * (x1 - L1)+ u1 * ((x1 - L1)**3) - k2 * (x1 - L2-x2)+ u2 *  (x1 - L2-x2)**3) / m1,
         y2,
         (-b2 * y2 - k2 * (x2 - x1 - L2) + u2 * (x2 - x1 - L2)**3 ) / m2]
    return f
```

```
L1 = 0.0
L2 = 0.0
# Friction coefficients
b1 = 0.0
b2 = 0.0
#Non-linear coefficients
u1=-1.0/6.0
u2=-1.0/10.0

# Initial conditions
# x1 and x2 are the initial dis
```

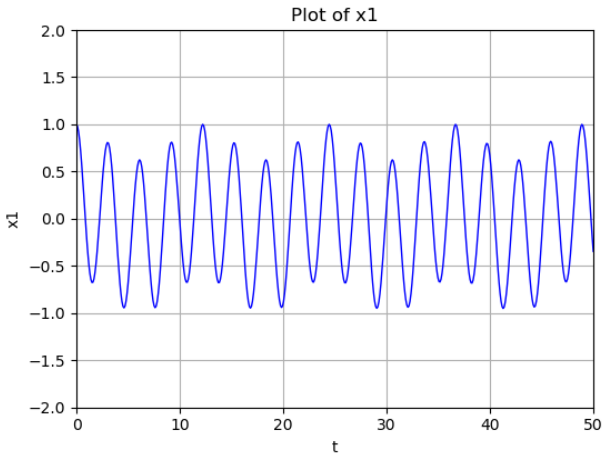
Y claramente, cada vez que se mencionara al vector p, se agregaron las variables u1 y u2.

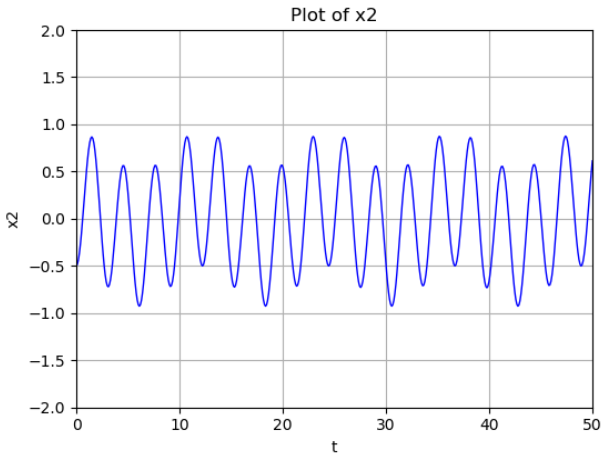
Las gráficas que se crearon, fueron las mismas que en los ejemplos anteriores, por lo que el código es análogo. Gráficas de fase, una para x_1 y otra para x_2.



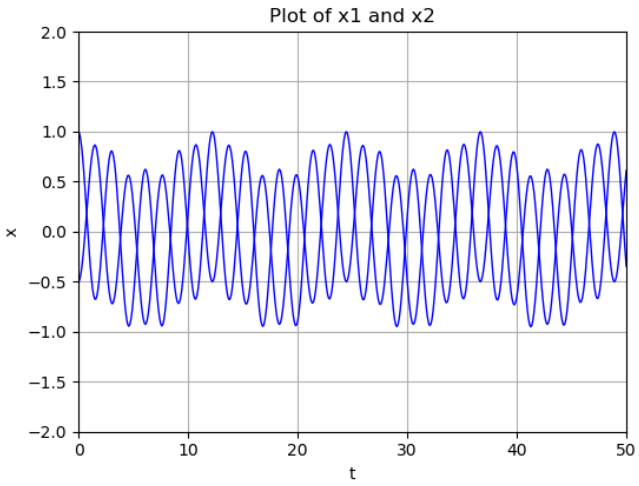


Plot de x_1 y x_2

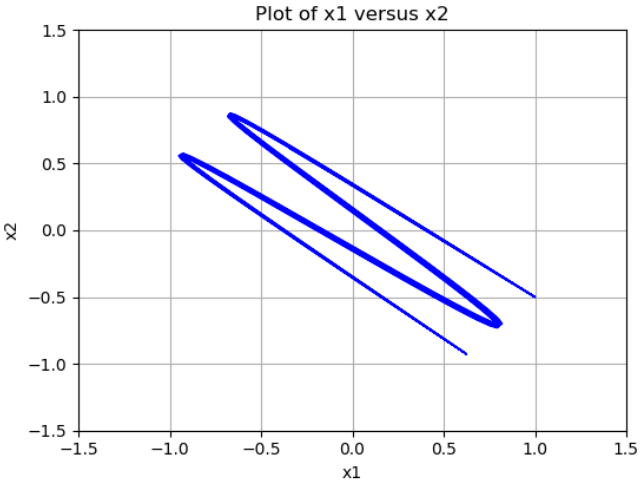




El movimiento de x_1 y x_2 con respecto al tiempo



Plot de x_1 vs x_2.

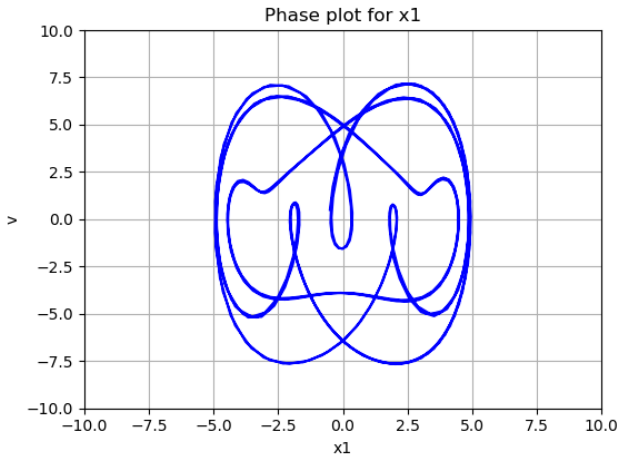


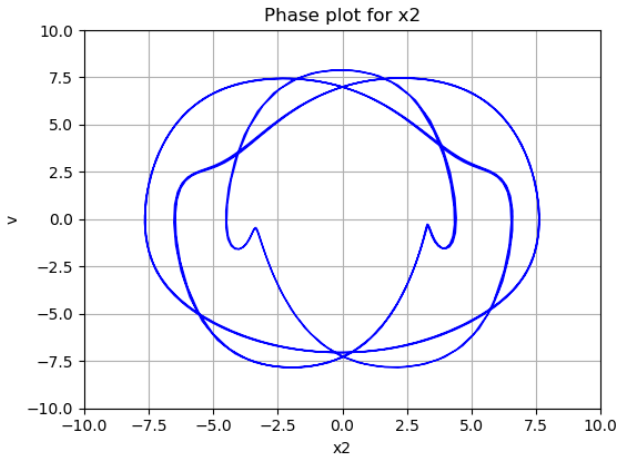
- **Ejemplo 3.2**

  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, δ_2= 0, coeficientes no lineales μ_1=-1/6, μ_2=-1/10 y condiciones iniciales $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0))$ = (-0.5, 1/2, 3.001, 5.9).

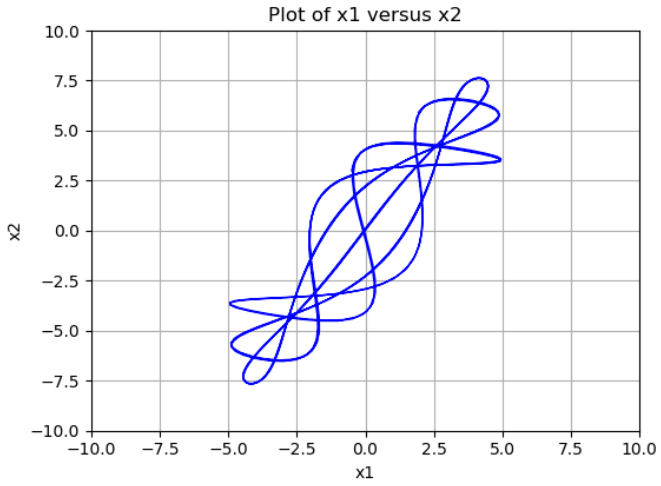
  – **Solución numérica con Python** El código utilizado fue el mismo, con pequeñas modificaciones en las condiciones iniciales. En esta ocasión se realizaron menos gráficas.

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





Plot de x_1 vs x_2.



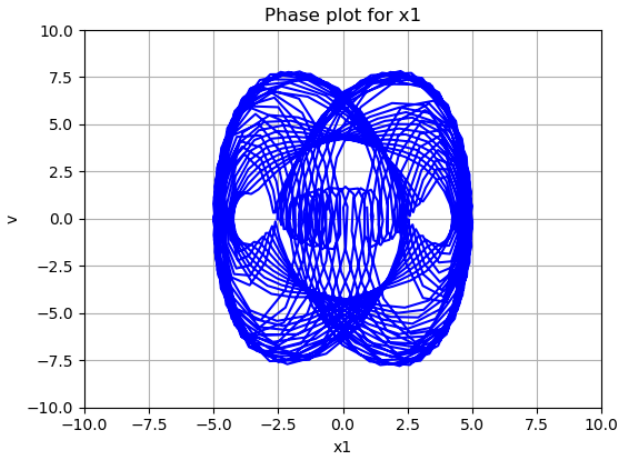
- **Ejemplo 3.3**

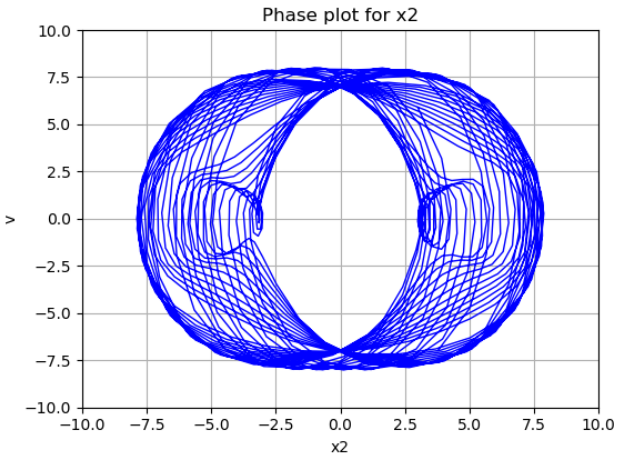
  Describir el movimiento de los resortes con $m_1 = m_2 = 1$, k_1=0.4, k_2=1.808, constantes de amortiguamiento δ_1= 0, δ_2= 0, coeficientes no lineales μ_1=-1/6, μ_2=-1/10 y condiciones iniciales $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0))$ = (-0.6, 1/2, 3.001, 5.9).

  En este caso y el anterior, vemos que únicamente variamos las condiciones iniciales y las gráficas resultan bastante diferentes.

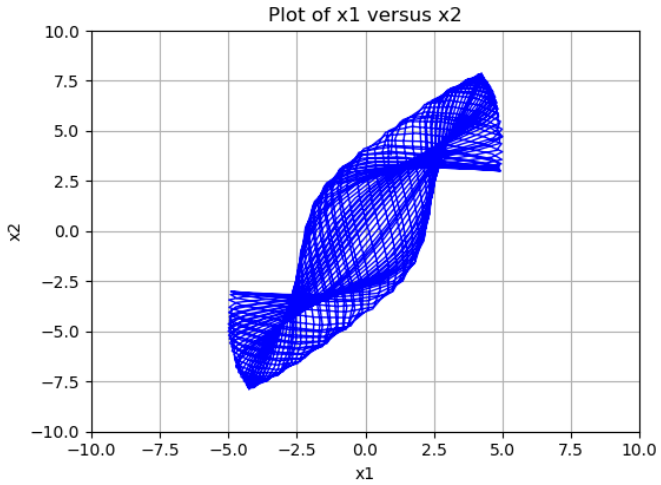
  – **Solución numérica con Python** Al igual que en el ejemplo anterior únicamente se realizaron tres gráficas.

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





Plot de x_1 vs x_2.



2.4 Agregando fuerzas externas

Agregar fuerzas externas es bastante sencillo, únicamente se tiene que añadir a las ecuaciones la fuerza que estemos agregando, en este caso suponemos la fuerza externa de manera periódica, sinusoidal de la forma $Fcos\omega t$. El modelo queda:

$$m_1\ddot{x}_1 = -\delta_1\dot{x}_1 - k_1x_1 + \mu_1x_1^3 - k_2(x_1 - x_2) + \mu_2(x_1 - x_2)^3 + F_1cos\omega_1 t \tag{7}$$

$$m_2\ddot{x}_2 = -\delta_2\dot{x}_2 - k_2(x_2 - x_1) + \mu_2(x_2 - x_1)^3 + F_2cos\omega_2 t \tag{8}$$

El movimiento que se genera al añadir una fuerza externa puede ser de diferentes tipos, puede resultar en una solución armónica o en una resonancia no lineal, entre otros, todo

depende de la relación entre la fuerza que se induzca y de la que ya tenía el resorte.

- **Ejemplo 4.1**

  Describir el movimiento de los resortes con $m_1 = m_2 = 1$, k_1=2/5, k_2=1, constantes de amortiguamiento δ_1= 1/10, δ_2= 1/5, coeficientes no lineales μ_1=1/6, μ_2=1/10, amplitudes de fuerza F_1= 1/3, F_2 =1/5, frecuencias de fuerza ω_1=1, ω_2=3/5 y condiciones iniciales $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (0.7,\ 0,\ 0.1,\ 0)$.

  Debido al amortiguamiento, cuando t es pequeño, el movimiento va a ser muy diferente a cuando t es grande, donde se estabiliza el comportamiento, por eso se mostrarán los limit cycles, que son gráficas de fase para valores de t mayores.

  – **Solución numérica con Python**

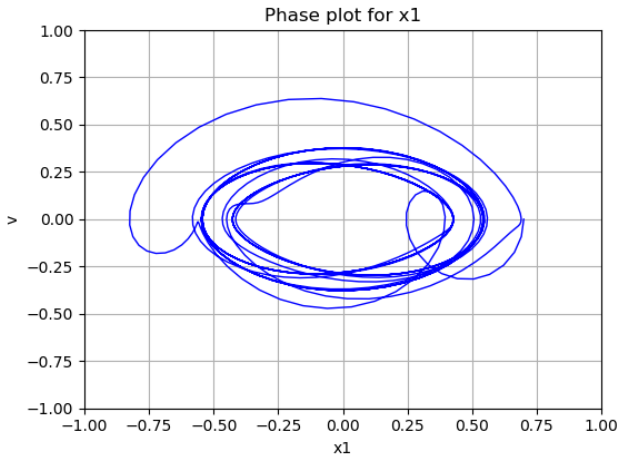
    De manera similar al ejemplo 3.1, se utilizó el mismo código que en el 2.1 y únicamente se muestran a continuación las modificaciones realizadas, tomando en cuenta que el vector p cambia también y se deben realizar los cambios pertinentes para que el código corra sin problemas.

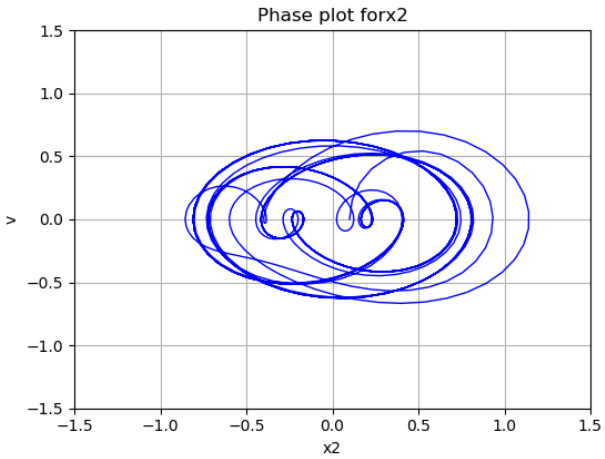
```
In [1]: import numpy as np
def vectorfield(w, t, p):
    """
    Defines the differential equations for the coupled spring-mass system.

    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]
    """
    x1, y1, x2, y2 = w
    m1, m2, k1, k2, L1, L2, b1, b2,u1,u2, f1,f2,q1,q2 = p

    # Create f = (x1',y1',x2',y2'):
    f = [y1,
         (-b1 * y1 - k1 * x1 + u1*(x1**3) -k2 * (x1-x2)+ u2*((x1-x2)**3)+ f1*(np.cos(q1*t)) ) / m1,
         y2,
         (-b2 * y2 - k2 * (x2 - x1)+ u2*((x2-x1)**3)+ f2*(np.cos(q2*t))) / m2]
    return f
```

```
u1=1.0/6.0
u2=1.0/10.0
#Forcing amplitudes
f1=1.0/3.0
f2=1.0/5.0
#Forcing frequencies
q1=1.0
q2=3.0/5.0
```

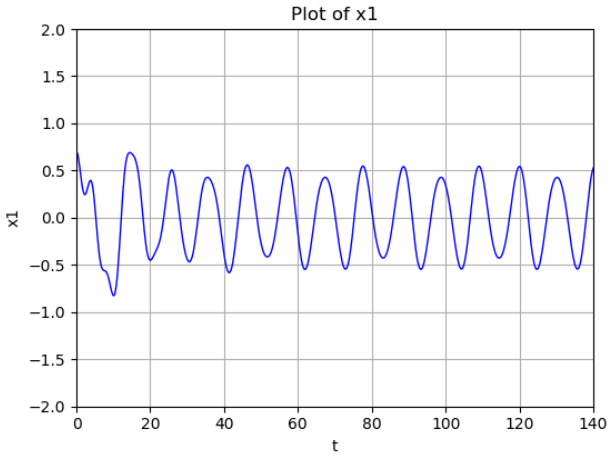
Las gráficas realizadas son las mismas que se han estado trabajando.

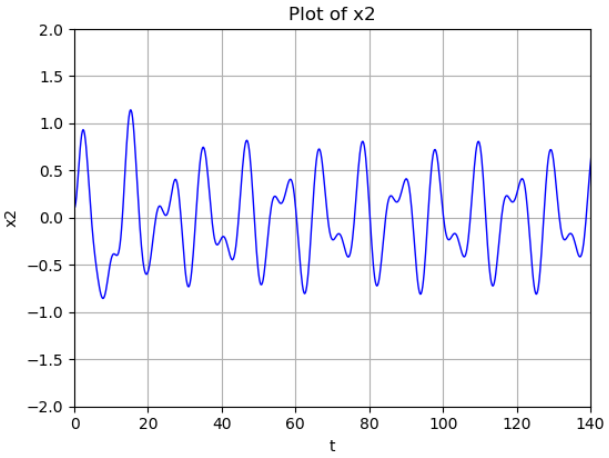
Gráficas de fase, una para x_1 y otra para x_2.





Plot de x_1 y x_2



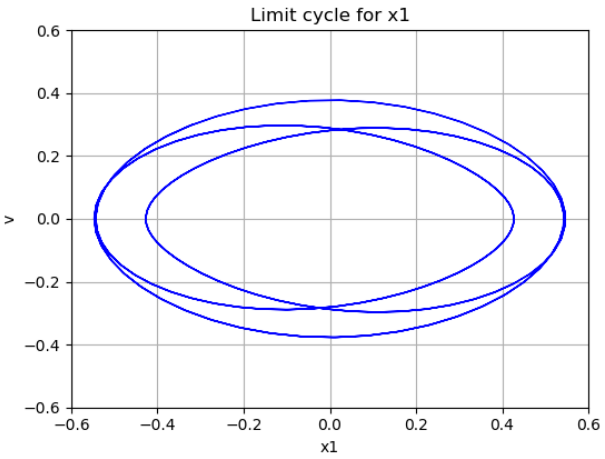


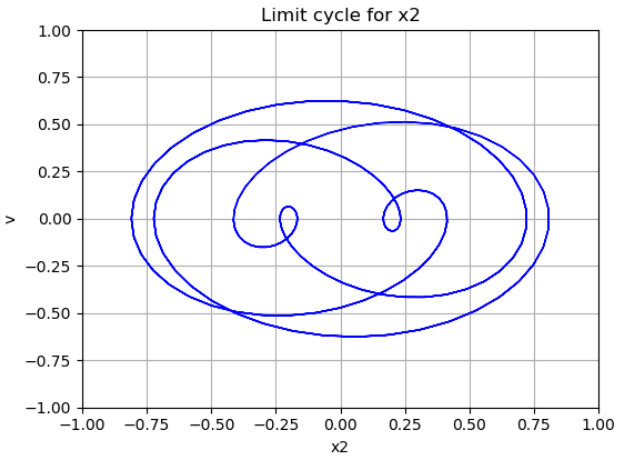
Como se mencionó, cuando se agrega fuerza, los primeros valores de t suelen mostrar alteraciones extrañas en el movimiento, mientras que para t más grande, se estabiliza. Para mostrar esto las siguientes gráficas utilizan t>110, mientras que las anteriores empiezan desde que t=0.

Para poder realizar esto, nos saltamos las primeras 550 líneas del archivo que creamos en un inicio:

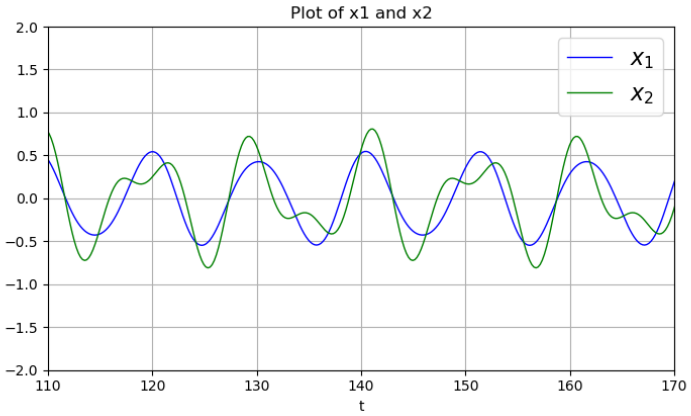
```
# Plot the solution that was generated

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 = loadtxt('ej4_1.dat', unpack=True, skiprows=550)

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

xlabel('x1')
ylabel('v')
grid(True)
#hold(True)
lw = 1
plt.xlim(-0.6,0.6)
plt.ylim(-0.6,0.6)


plot(x1,y1, 'b', linewidth=lw)

#Legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Limit cycle for x1')
savefig('ej4_1.5.png', dpi=100)
```

Así pues creamos limit cycles (gráficas de fase):

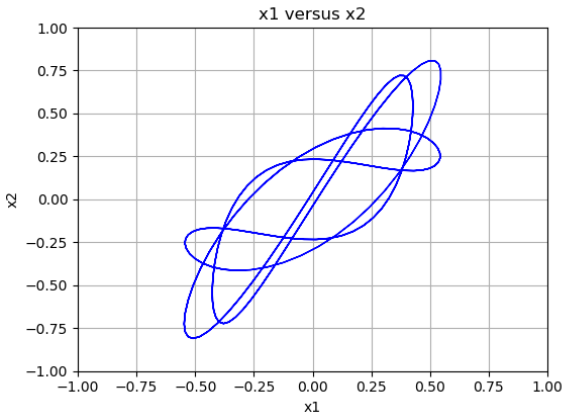




El movimiento de x_1 y x_2 con respecto al tiempo



Plot de x_1 vs x_2.



3 Conclusiones

Agregando a las conclusiones de la semana pasada, reitero que me agradó mucho el ambiente de Jupyter Lab, bastante completo.

Por otro lado, al respecto del artículo, creo que esta semana se puso más interesante, se volvió un mayor reto, puesto que la semana pasada únicamente nos dedicamos a entender el código y modificar pequeñas partes, esta semana si cambiamos algo significativo y se creó un código que es flexible para cualquiera de los casos mencionados con anterioridad.

Me llamó mucho la atención que las gráficas resultantes fueran idénticas a los modelos analíticos, lo que nos dice que estos modelos nos dan una aproximación bastante buena.

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

   Me gustó mucho que nosotros pudieramos modificar y analizar el código para acomodarlo a nuestras necesidades.

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?

   No, nunca me hubiera imaginado esto, puesto que en clase cuando analizamos este tipo de sistemas siempre nos vamos a lo más idealizado y no profundizamos al respecto.

3. ¿Qué propondrías para mejorar esta actividad? ¿Te ha parecido interesante este reto?

   Me parece que ya tiene lo necesario, aunque siempre realizamos lo mismo, las pequeñas variaciones nos permiten dar el análisis físico.

4. ¿Quisieras estudiar mas este tipo de fenómenos no lineales?

   Si me parece interesante, sin embargo, por el momento no me encuentro muy familiarizada al respecto.