Actividad número 7: Sistema de resortes acoplados (continuación actividad 6)

Juan Pedro Barajas Ibarria

19 de marzo de 2018

Ecuaciones de resortes acoplados [3]

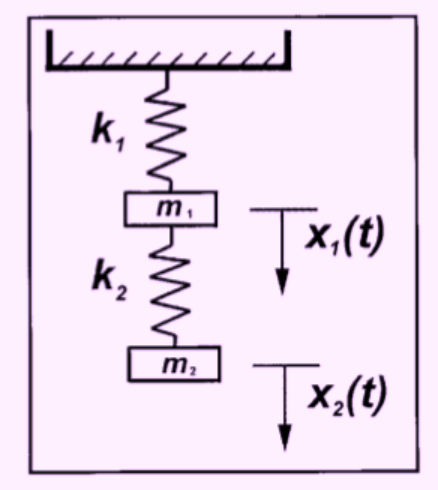
Las ecuaciones de sistemas de resortes acoplados para describir el movimiento de dos resortes con masas conectados entre si. Para el modelo lineal que usa la ley de Hooke, el movimiento de cada peso se describe por una ecuación diferencial de cuarto orden. También se describe un modelo no lineal y se consideran la amortización y el forzamiento externo.

1. Introducción

Bajo la suposición de que las fuerzas de restauración se comportan de acuerdo con la Ley de Hooke, este problema de dos grados de libertad se modela mediante un par de ecuaciones diferenciales lineales de segundo orden acopladas. Al diferenciar y suponer una ecuación en la otra, se puede demostrar que el movimiento de cada peso está determinado por una ecuación diferencial lineal de cuarto orden.

2. El modelo de resorte acoplado

El modelo consiste en dos resortes y dos pesos. Un resorte, que tiene una constante de resorte k_1, está unido al techo y un peso de masa m_1 está unido al extremo inferior de este resorte. A este peso, se une un segundo resorte que tiene una constante de resorte k_2. En la parte inferior de este segundo resorte, se pone un peso de masa m_2 y todo el sistema aparece como se ilustra en la figura siguiente.



2.1. Asumiendo la ley de Hooke

Bajo condición asumida de oscilaciones pequeñas, las fuerzas de restauración son de la forma $-k_1 l_1$ y $-k_2 l_2$ donde l_1 y $l_[2]$ son el alargamiento (o compresión) de los dos resortes. Desde que la masa es agregada a los resortes, ellos tendrán una fuerza de restauración actuando sobre ellas. Una fuerza de restauración $-k_1 x_1$ ejercido por el alargamiento (o compresión) x_1 del primer resorte. y la fuerza hacia arriba $-k_2(x_2 - x_1)$ del segundo resorte. Si asumimos que no hay forzamiento, entonces la segunda ley de Newton implica dos ecuaciones que representan el movimiento de los dos pesos son

$$\begin{aligned} m_1 \ddot{x}_1 &= -k_1 x_1 - k_2(x_1 - x_2) \\ m_2 \ddot{x}_2 &= -k_2(x_2 - x_1) \end{aligned} \tag{1}$$

Así tenemos un par de ecuaciones diferenciales lineales acopladas de segundo orden. Para encontrar una ecuación para x_1 que no implique x_2, resolvemos la primera ecuación para x_2, así sustituyéndola en ecuaciones anteriores se tiene una ecuación de cuarto grado la cual ya se puede resolver

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

Entonces solo se necesitaran las condiciones iniciales para la solución a la ecuación.

2.2. Algunos ejemplos con masas iguales

Consideramos que el modelo es con dos masas de igual masa. Esto sera con $m_1 = m_2 = 1$. En el caso de no tener forzamiento tenemos la ecuación característica que es

$$m^4 + (k_1 + 2k_2)m^2 + k_1k_2 = 0$$

Ejemplo 2.1 Se describe el movimiento de un sistema de resortes con $k_1 = 6$ y $k_2 = 4$ con las condiciones iniciales $(x_1(0), \dot{x_1}(0), x_2(0), \dot{x_2}(0)) = (1, 0, 2, 0)$. De forma analítica, se llega a que las ecuaciones de posición son:

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

El movimiento es sincronizado, y por tanto las masas se mueven en fase, esto es una con respecto . a la otra. Solo difieren en amplitud. La gráfica de la fase de estos movimientos son curvas cerradas (elipses) , ademas como solo cambian en amplitud, al gratificarlas una contra la otra se tiene una recta. El código que se utilizo para resolver la ecuación de manera numérica fue recuperado de la pagina de SciPy Cookbook [1] , donde se utilizaron la función de NumPy: scipy.integrate.odeint [2]. Para resolver las ecuaciones, a continuación de muestra la variante del código adaptado a nuestro caso donde se cambiaron los parámetros y las condiciones iniciales. Primero se genero el vector para la solución numérica donde estarán las ecuaciones.

```
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
```

Después cambiando los parámetros resolvemos y generamos los archivos de datos para las representaciones gráficas.

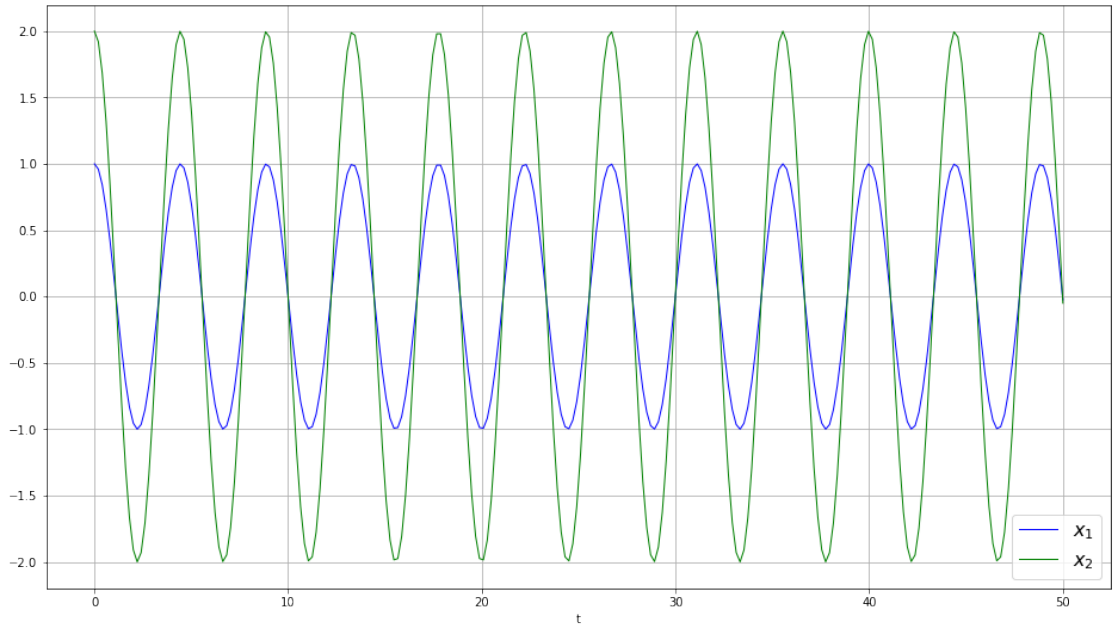
In [2]:
```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 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('two_springs.dat', 'w') as f:
```

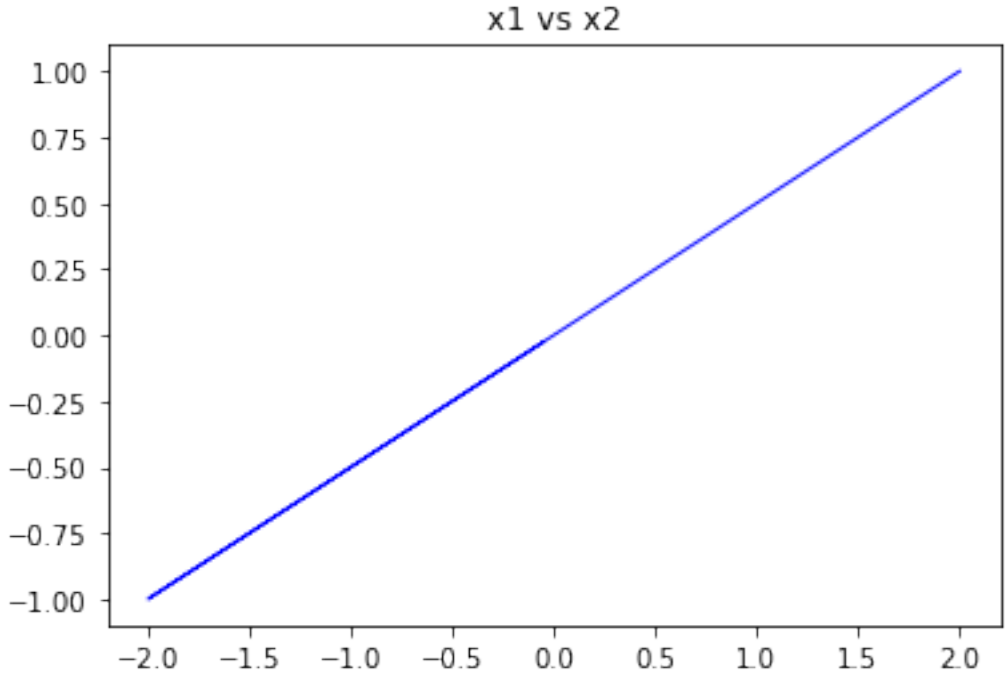
```
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print ( t1, w1[0], w1[1], w1[2], w1[3],file=f)
```

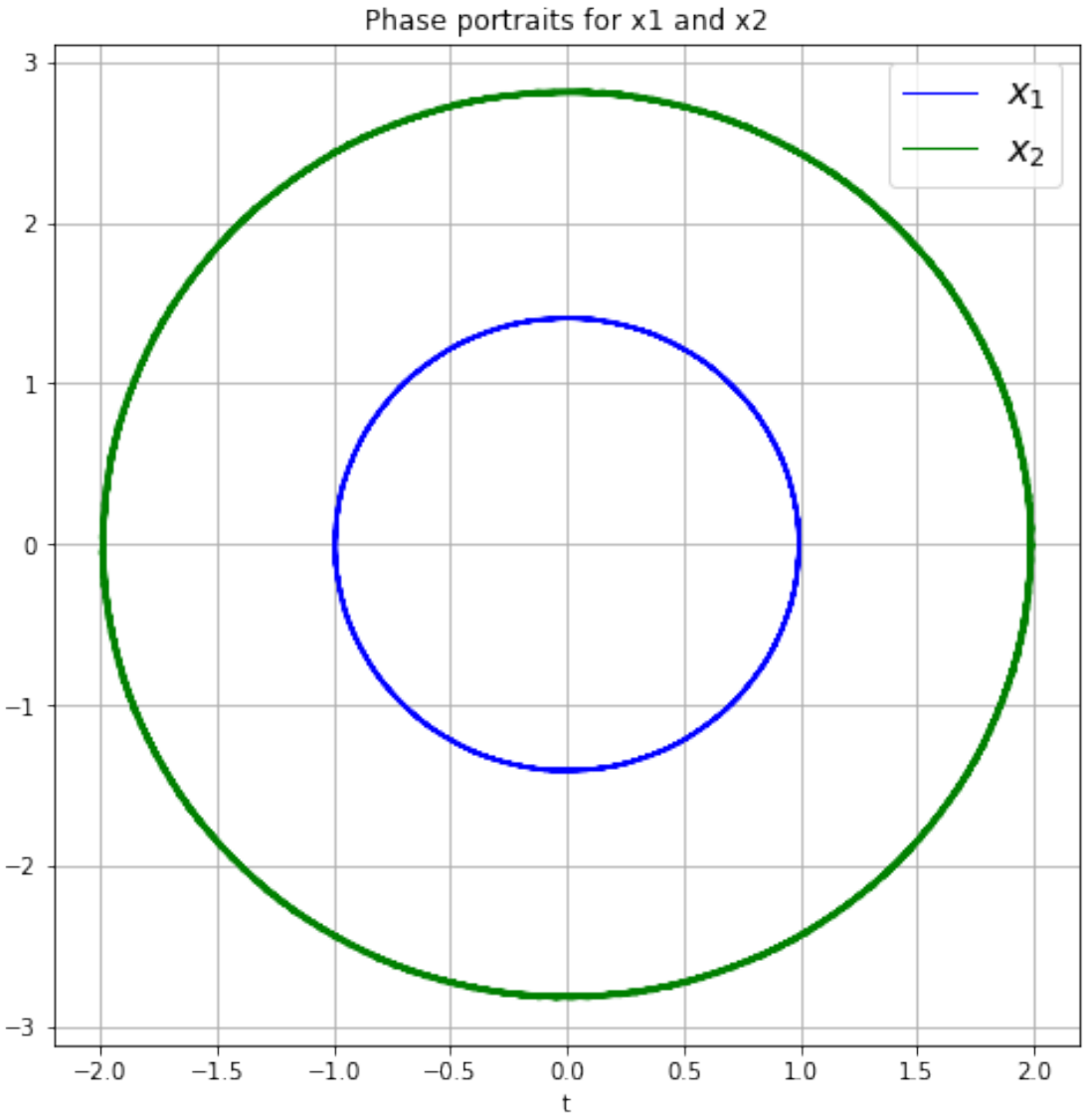
Las gráficas de la solución para el caso particular son las siguientes donde el código para graficar las soluciones, viene incluido el titulo de cada imagen en ella.

In [3]:
```
# Plot the solution that was generated

from numpy import loadtxt
from pylab import figure, plot, xlabel, grid, hold, legend, title, savefig
from matplotlib.font_manager import FontProperties
%matplotlib inline
t, x1, xy, x2, y2 = loadtxt('two_springs.dat', unpack=True)

figure(1, figsize=(16, 9))

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

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

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
#title('')
savefig('two_springs2.1.png', dpi=100)
```

Las gráficas generadas fueron:


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
10
20
30
40
50
t
x_1
x_2


x1 vs x2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0



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

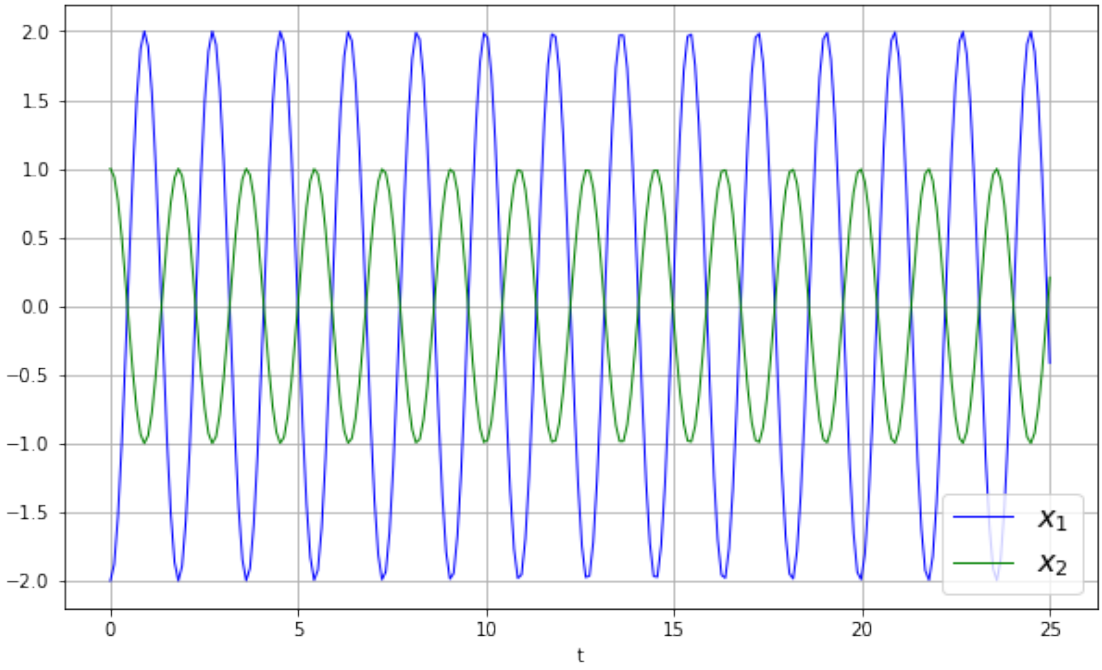
$$x_1(t) = -2\cos 2\sqrt{3}t$$
$$x_2(t) = \cos 2\sqrt{3}t$$

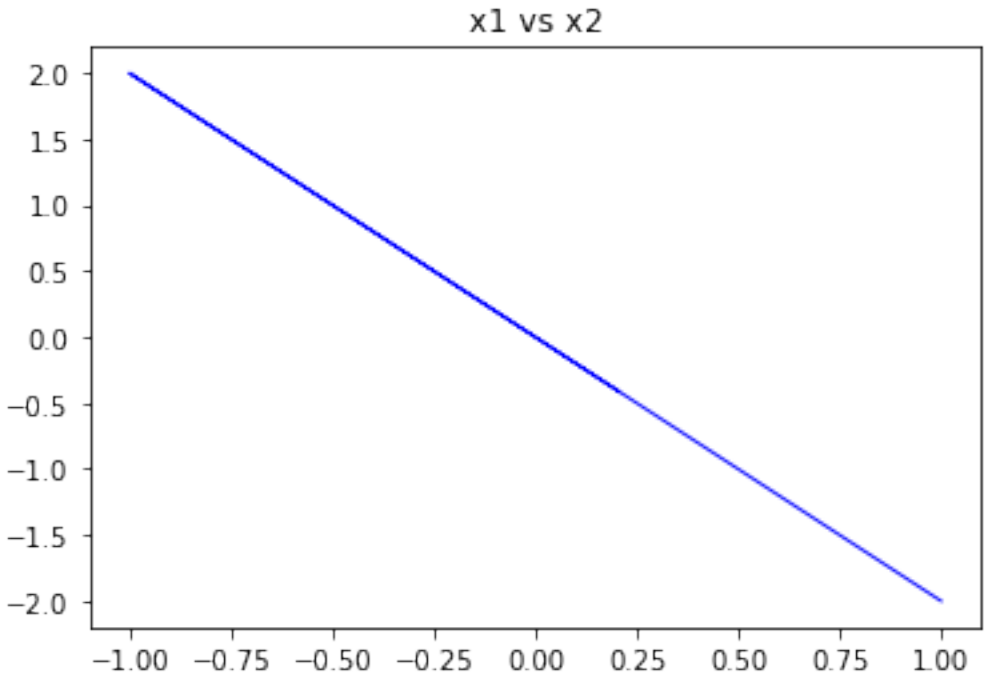
Donde usamos el mismo código que en el ejemplo anterior, simplemente cambiamos los parámetros y las condiciones iniciales donde se genero otro archivo para poder graficar de igual manera.

```
In [6]: # 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 = 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('two_springs2.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)
```

Las gráficas generadas fueron:



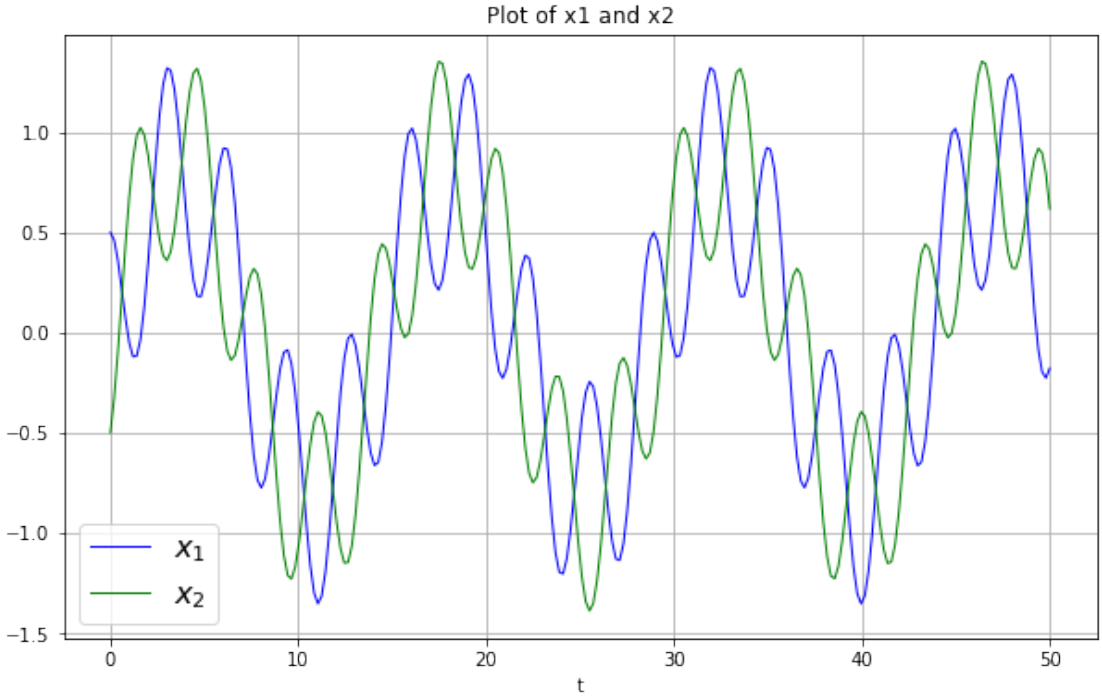


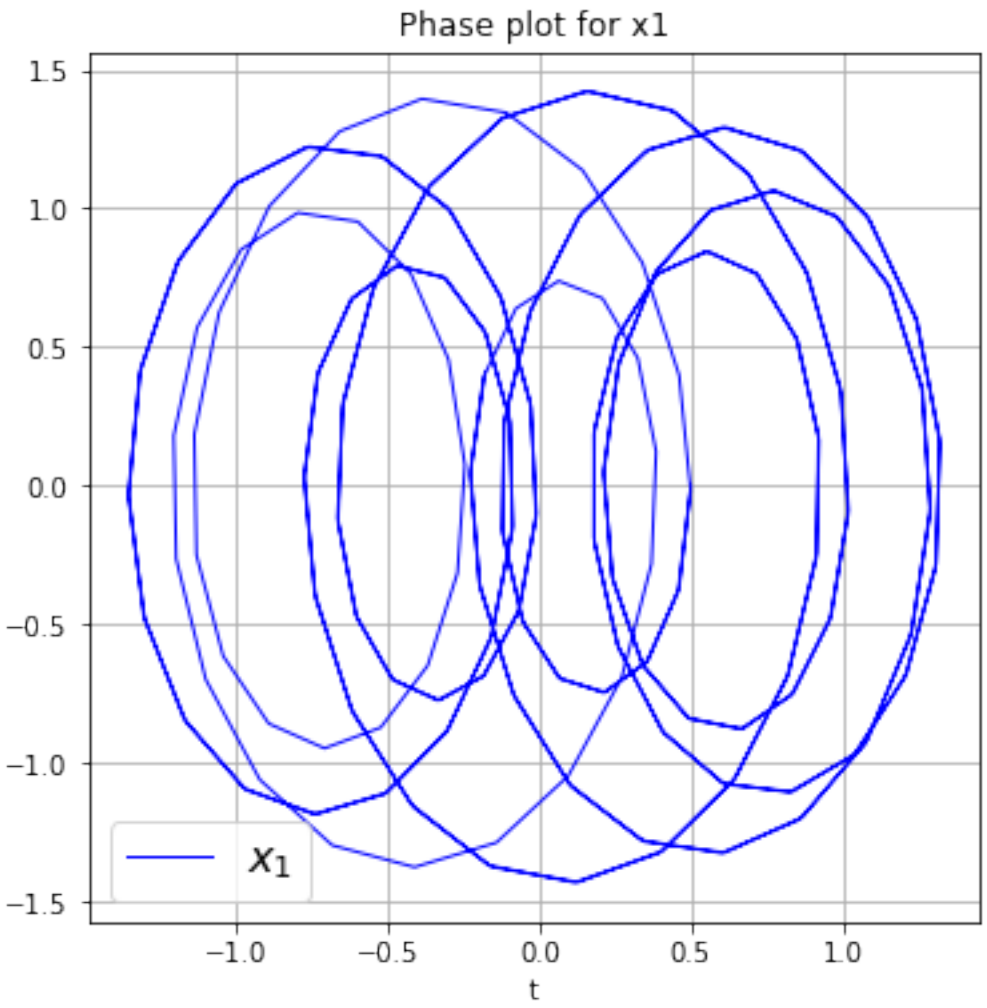
Ejemplo 2.3 Describe el movimiento para un sistema de resortes con $k_1 = 0.4$ y $k_2 = 1.808$ con condiciones iniciales $(x_1(0), \dot{x_1}(0), x_2(0), \dot{x_2}(0)) = (1/2, 0, -1/2, 7/10)$. De igual manera el procedimiento fue el mismo utilizando el mismo código, solo cambiando los parámetros y las condiciones iniciales.

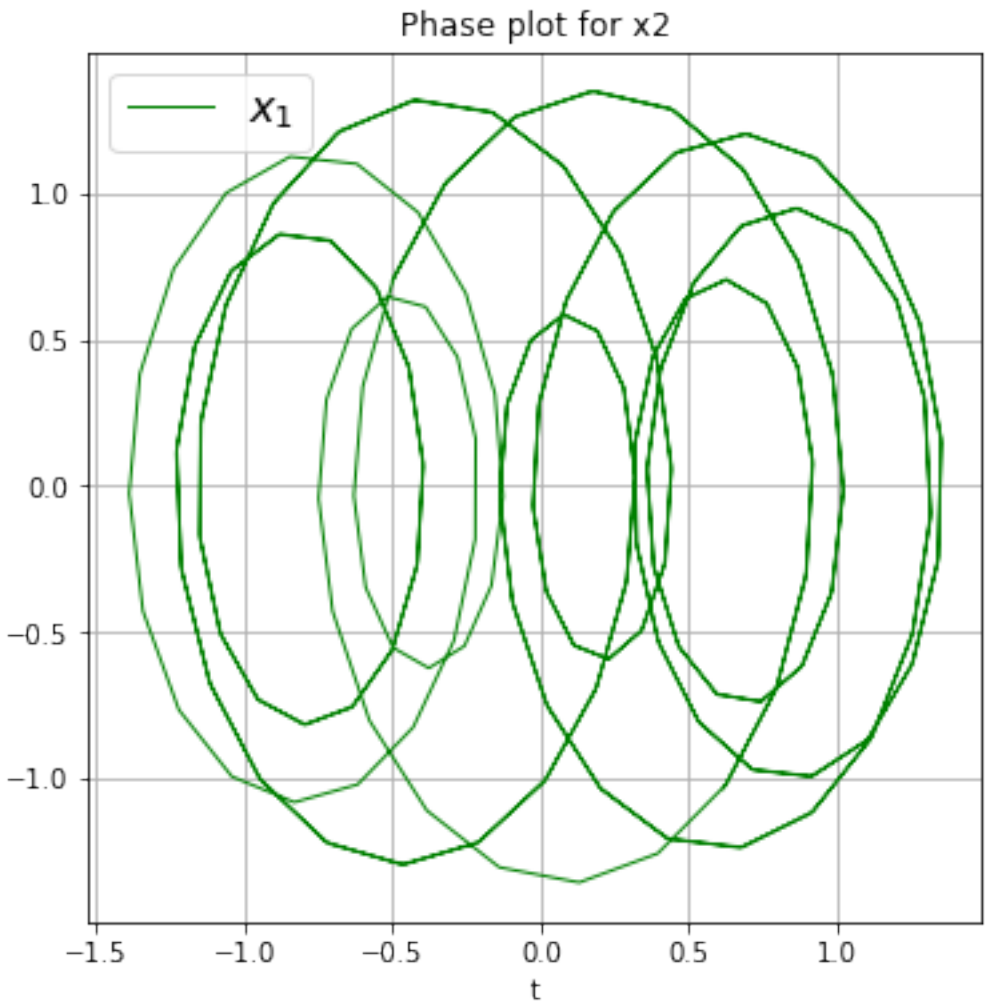
In [9]:
```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint

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

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = 1.0/2.0
y1 = 0.0
x2 = -1.0/2.0
y2 = 7.0/10.0

# 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.
```

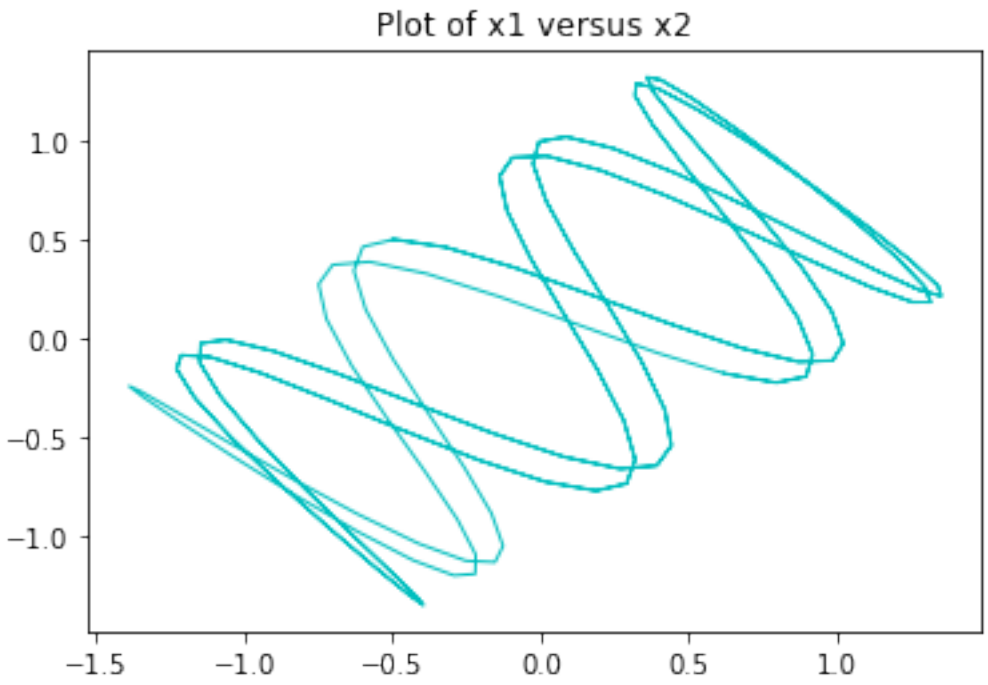
```
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

with open('two_springs3.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)
```

Las gráficas generadas fueron:




Phase plot for x1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
t
x_1


Phase plot for x2
x_1
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
t



2.3. Amortiguamiento

Los tipos de amortiguamiento mas comunes sobre amortiguamiento son los de viscosidad, donde la fuerza de amortiguamiento es proporcional a la velocidad. El amortiguamiento de la masa 1 depende solamente de su velocidad y no de la segunda. Si se añade esta fuerza al modelo anterior se puede obtener:

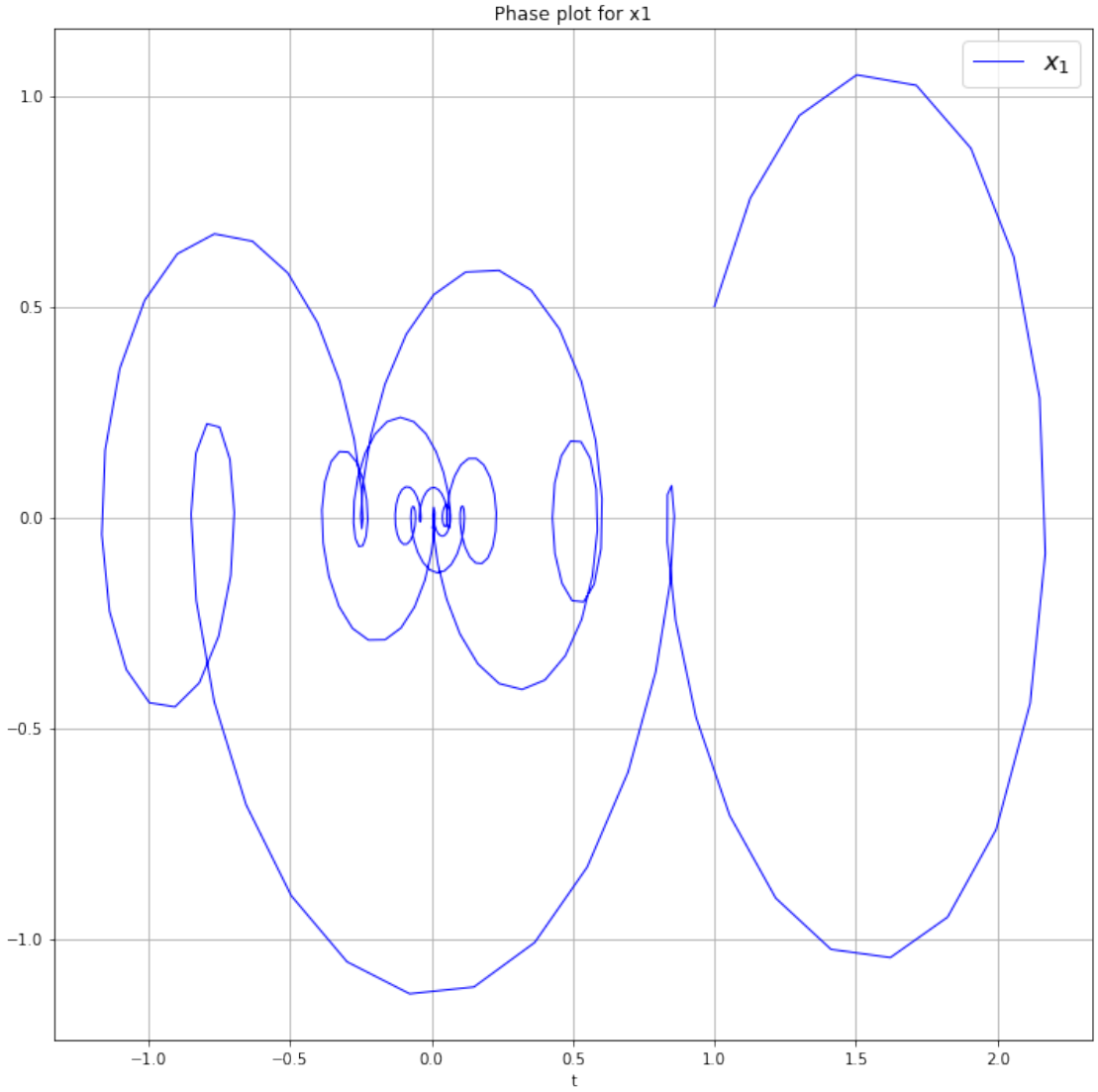
$$m_1\ddot{x}_1 = -\delta\dot{x}_1 - k_1x_1 - k_2(x_1 - x_2)$$
$$m_2\ddot{x}_2 = -\delta\dot{x}_2 - k_2(x_2 - x_1)$$

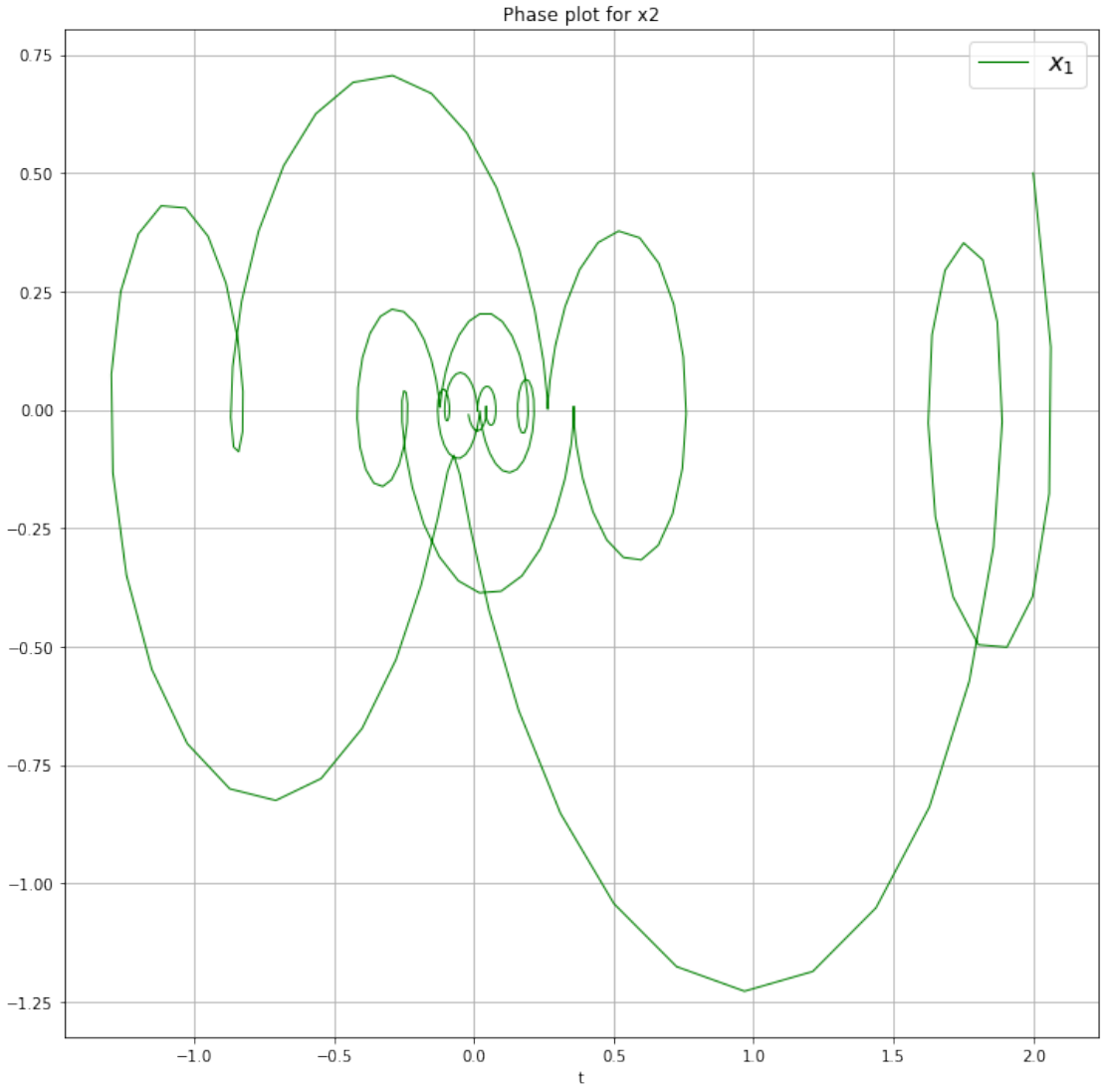
Si se realiza el mismo proceso para resolverla llegamos de igual forma al anterior sistema a una ecuación diferencial lineal de cuarto grado. *Ejemplo 2.4* Asume $m_1 = m_2 = 1$ Describe el movimiento de un sistema de resortes con $k_1 = 0.4$ y $k_2 = 1.808$, con coeficientes de amortiguamiento $\delta_1 = 0.1$ y $\delta_2 = 0.2$ con condiciones iniciales $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (1, 1/2, 2, 1/2)$. Para la obtención numérica de este resultado se modifico la ecuación a resolver añadiendo el factor de amortiguamiento y de igual manera a los ejemplos anteriores se modificaron los parámetros y las condiciones iniciales.

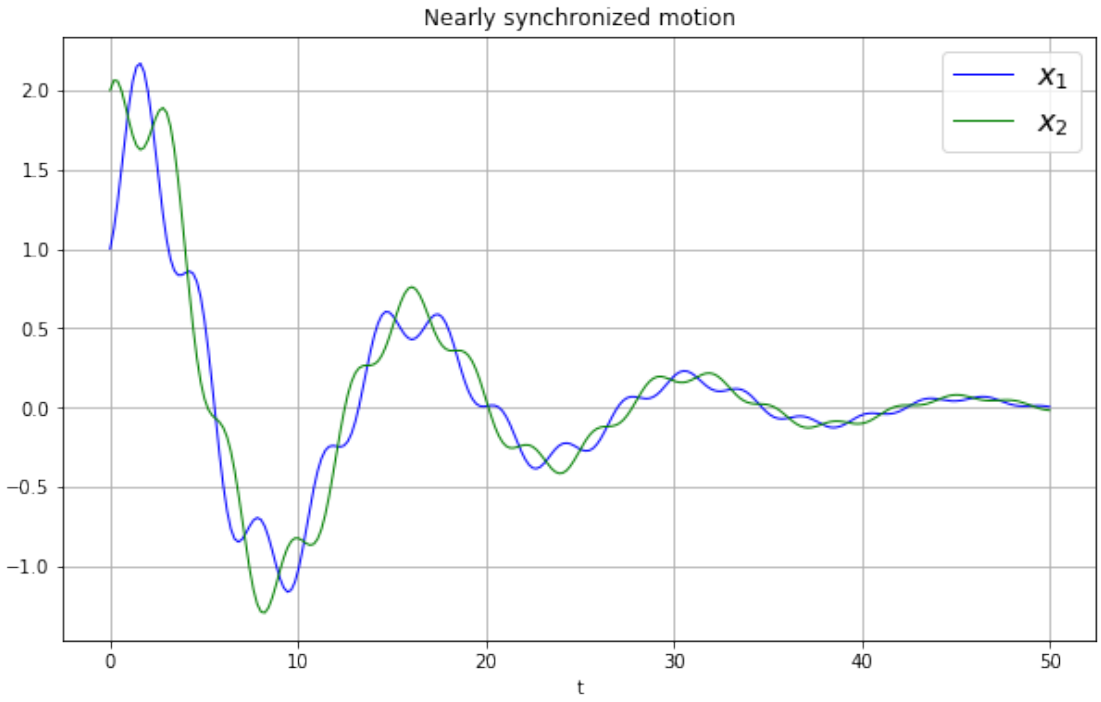
```
In [14]: # Use ODEINT to solve the differential equations defined by the vector field
         from scipy.integrate import odeint

         # Parameter values
```

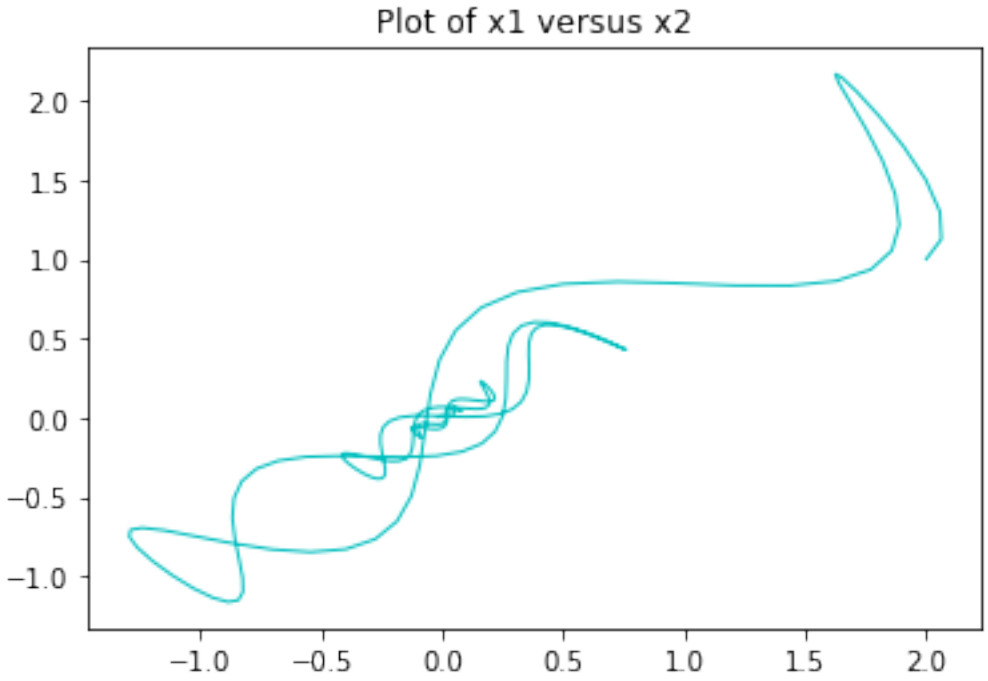
```
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 0.4
k2 = 1.808
# Natural lengths
L1 = 0.0
L2 = 0.0
# Friction coefficients
b1 = 0.1
b2 = 0.2

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

# 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('two_springs4.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)
```

Las gráficas obtenidas fueron las siguientes:


Phase plot for x1
x_1
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
t


Phase plot for x2
x_1
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
t





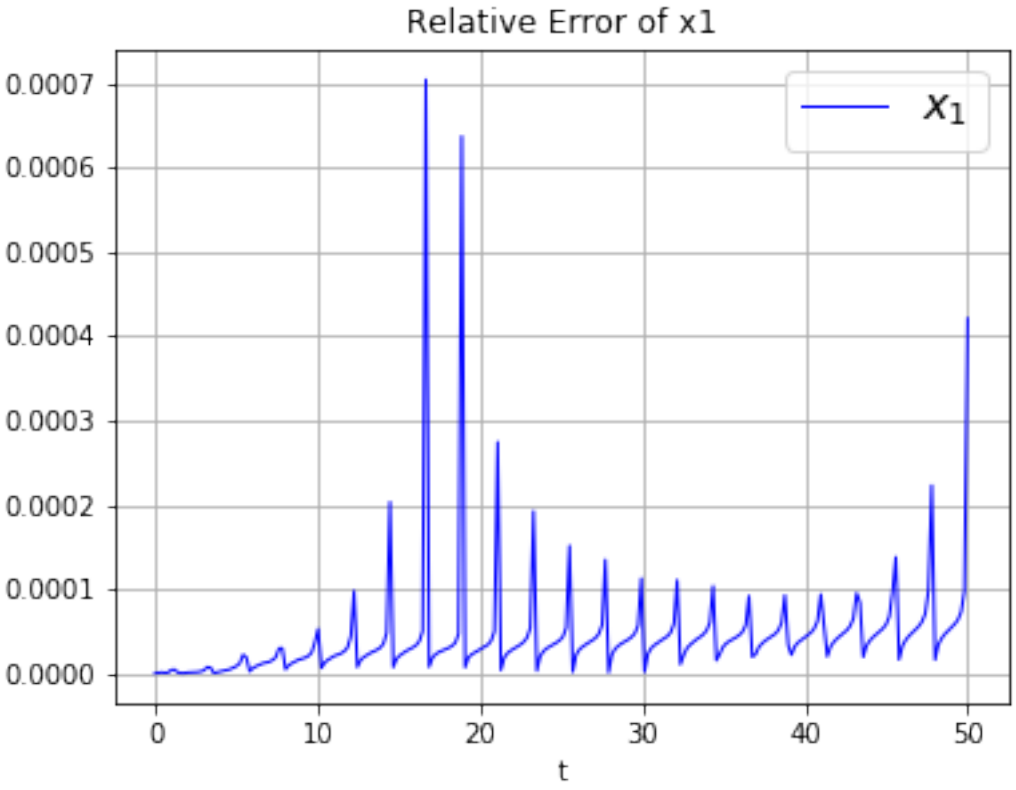
Calculo de errores

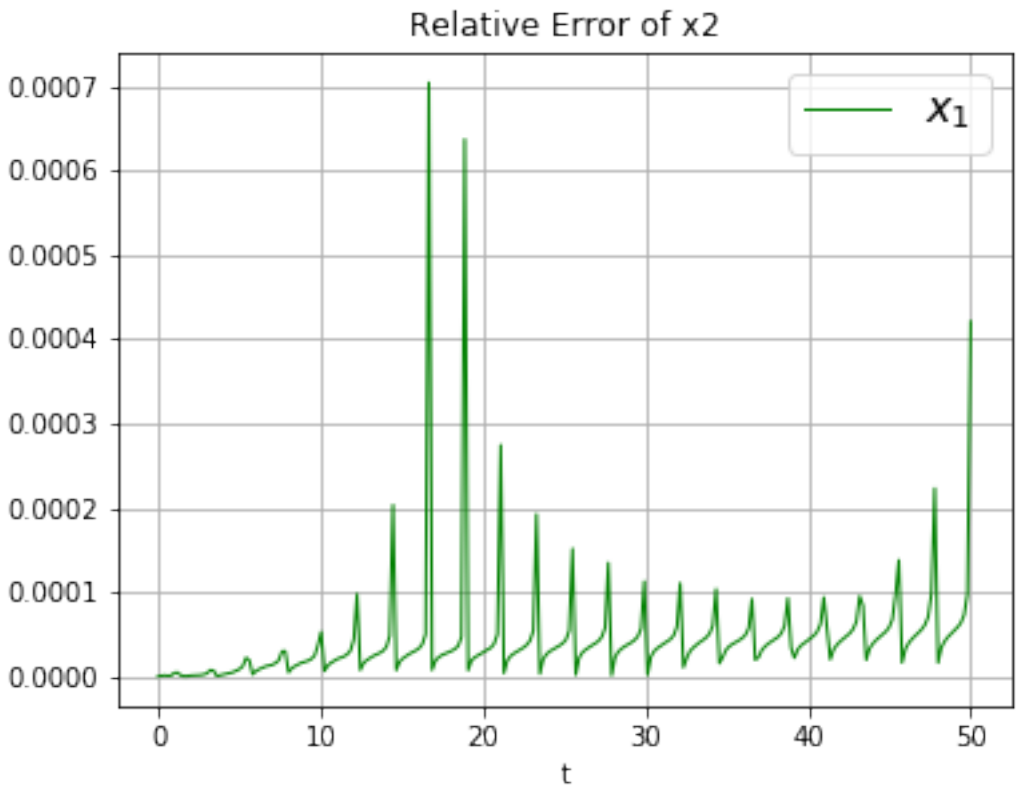
Para la ultima parte de la actividad se calcularon los errores absolutos entre las soluciones analíticas y las soluciones numéricas obtenidas en esta misma actividad. Esto

recreando las mismas situaciones, solamente al momento de guardar datos se calculo la diferencial con la solución real con cada ejemplo en particular.

In [12]:
```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
import numpy as np
import math

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 6.0
k2 = 4.0
# Natural lengths
L1 = 0
L2 = 0
# Friction coefficients
b1 = 0
b2 = 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.
```

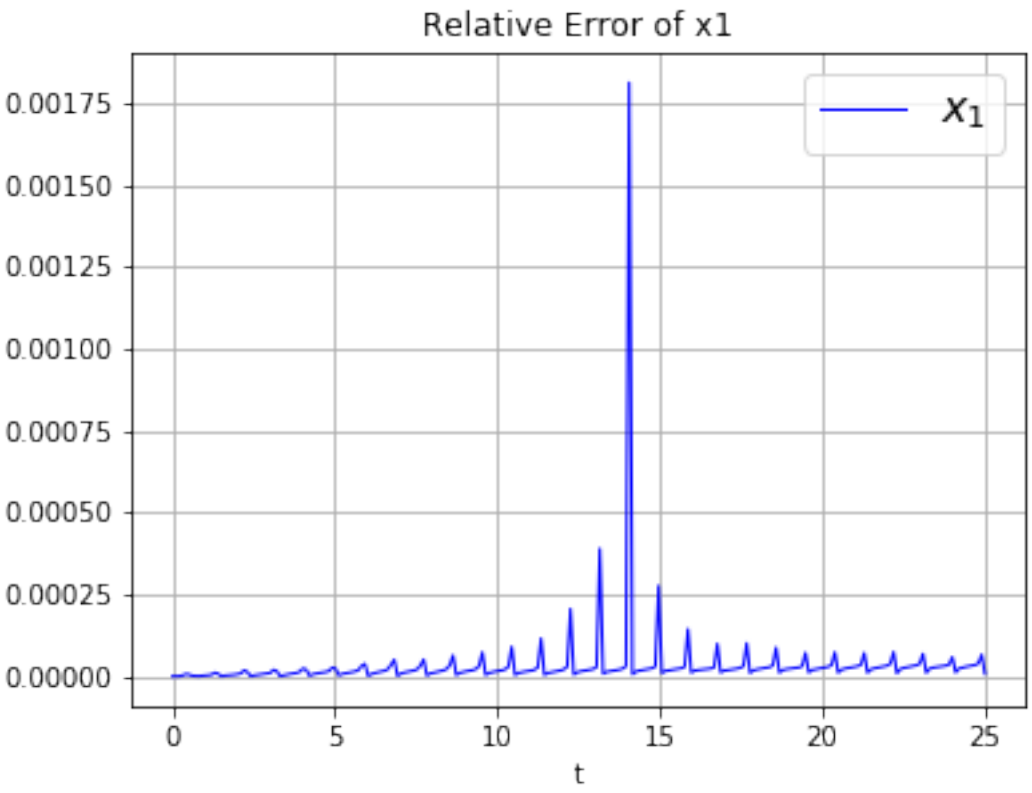
```
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

with open('datoserror_2_1.dat', 'w') as f:
    # Print & save the solution with absolut mistake.
    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)*t
```

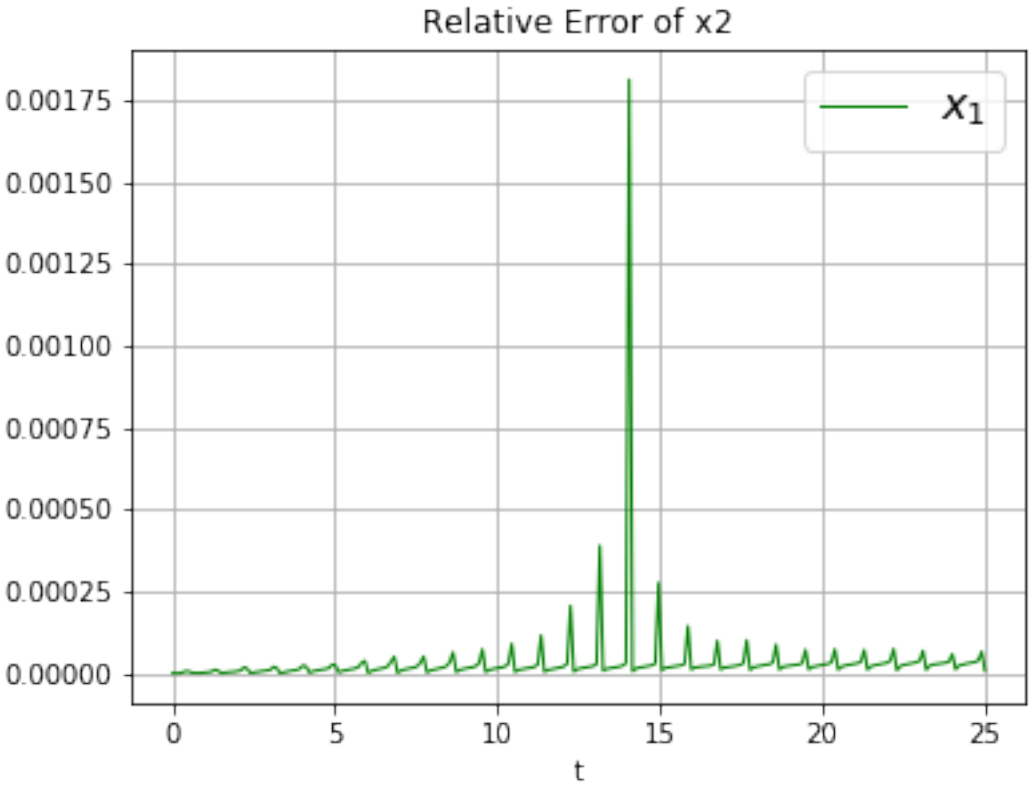




In [15]:
```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
import numpy as np
import math

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 6.0
k2 = 4.0
# Natural lengths
L1 = 0
L2 = 0
# Friction coefficients
b1 = 0
b2 = 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('datoserror_2_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
```


Relative Error of x1
x_1
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
0
5
10
15
20
25
t


Relative Error of x2
x_1
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
0
5
10
15
20
25
t

3. Añadiendo no linealidad

Si asumimos que la fuerza de restauración es no lineal, es decir, ya no es de la forma $-kx$, ahora suponemos que la fuerza es de la forma $-kx + \mu x^3$. Entonces nuestro modelo sera

$$m_1\ddot{x}_1 = -\delta_1\dot{x}_1 - k_1x_1 + \mu_1x_1^3 - k_2(x_1 - x_2) + \mu_2(x_1 - x_2)^3$$
$$m_2\ddot{x}_2 = -\delta_2\dot{x}_2 - k_2(x_2 - x_1) + \mu_2(x_2 - x_1)^3$$

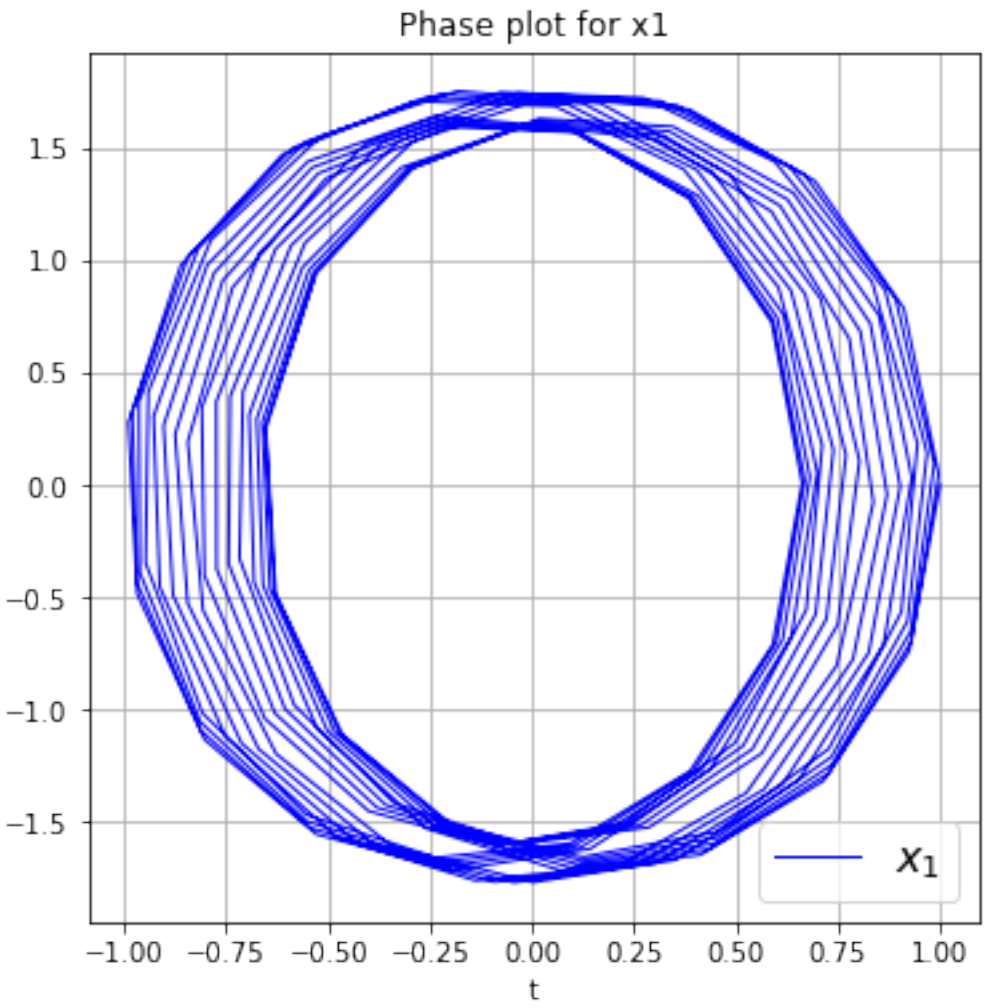
Al ser este modelo mas complicado por la no linealidad la exactitud se va perdiendo conforme va pasando el tiempo, el resolver las ecuaciones es mucho mas difícil por la misma razón.

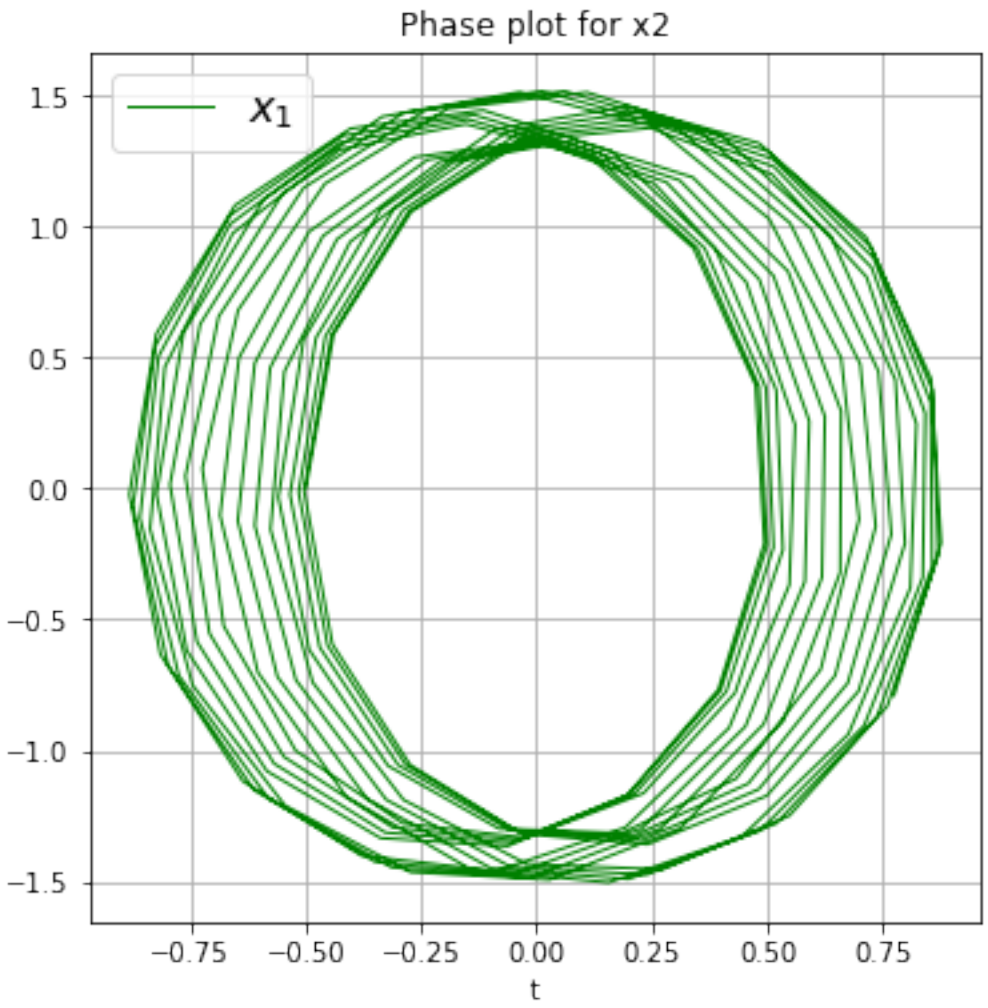
Ejemplo 3.1 Asumiendo $m_1 = m_2 = 1$. Describe el movimiento para un modelo de dos resortes con constantes $k_1 = 0.4$ y $k_2 = 1.808$, coeficientes de amortiguamiento $\delta_1 = 0$ $\delta_2 = 0$, coeficientes no lineales $\mu_1 = 1/6$ y $\mu_1 = -1/10$, con condiciones iniciales $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (1, 0, -1/2, 0)$. Aquí se implemento el mismo proceso de modificar las ecuaciones añadiendo el factor de no linealidad y resolviendo con el mismo código ya implementado simplemente modificando parámetros y condiciones iniciales.

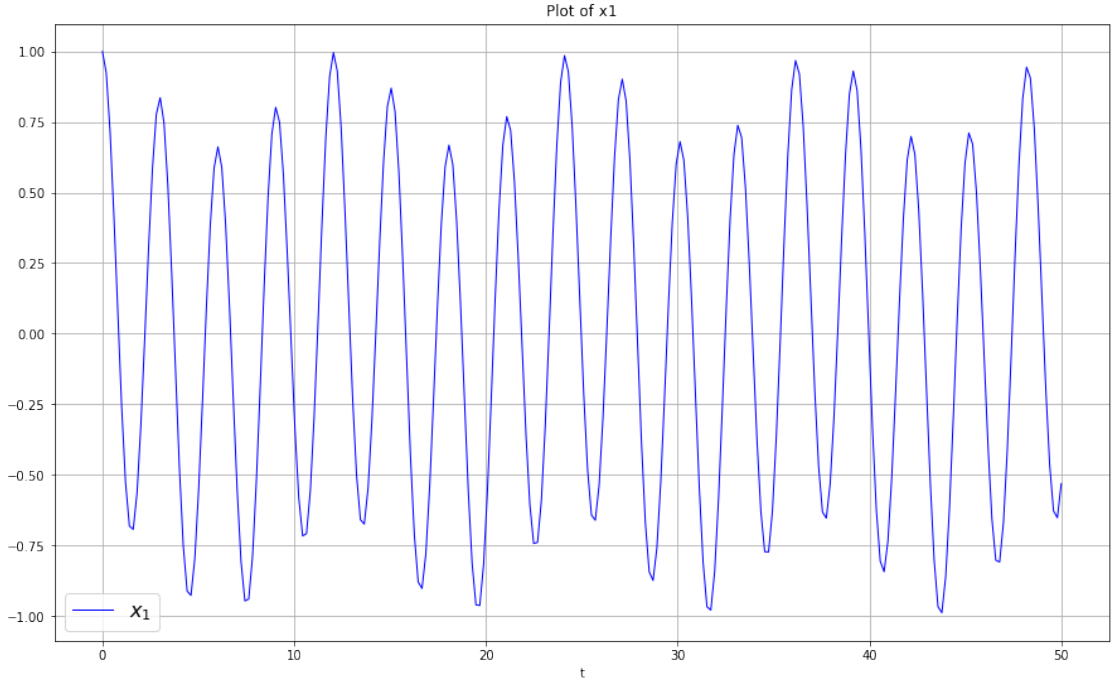
In [2]:
```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint

# 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
#nonlinearity coefficients
z1 = -(1.0/6.0)
z2 = -(1.0/10.0)

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

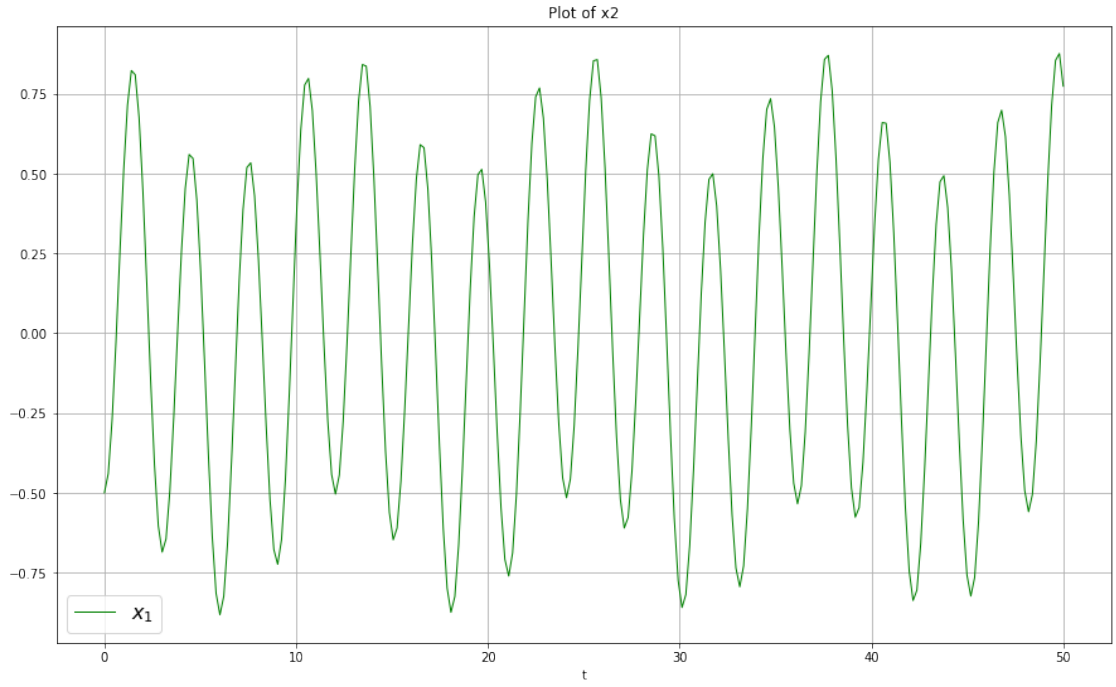
```
# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50.0
numpoints = 250

# 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, z1, z2 ]
w0 = [x1, y1, x2, y2]

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

with open('two_springsnl.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)
```

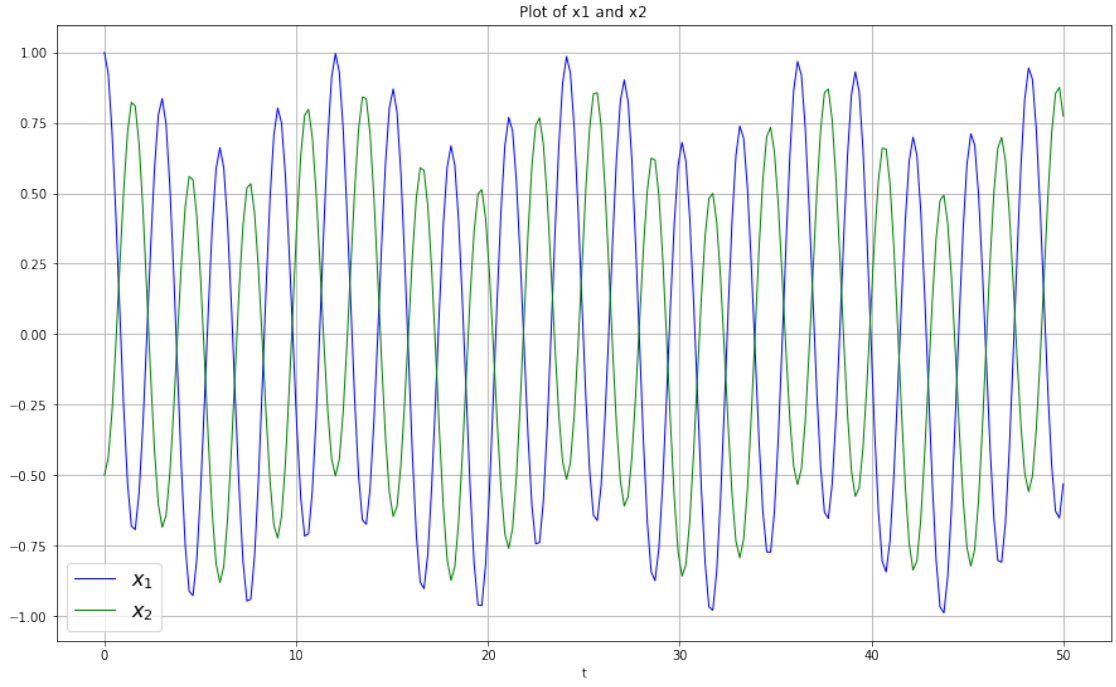
Las gráficas obtenidas fueron:

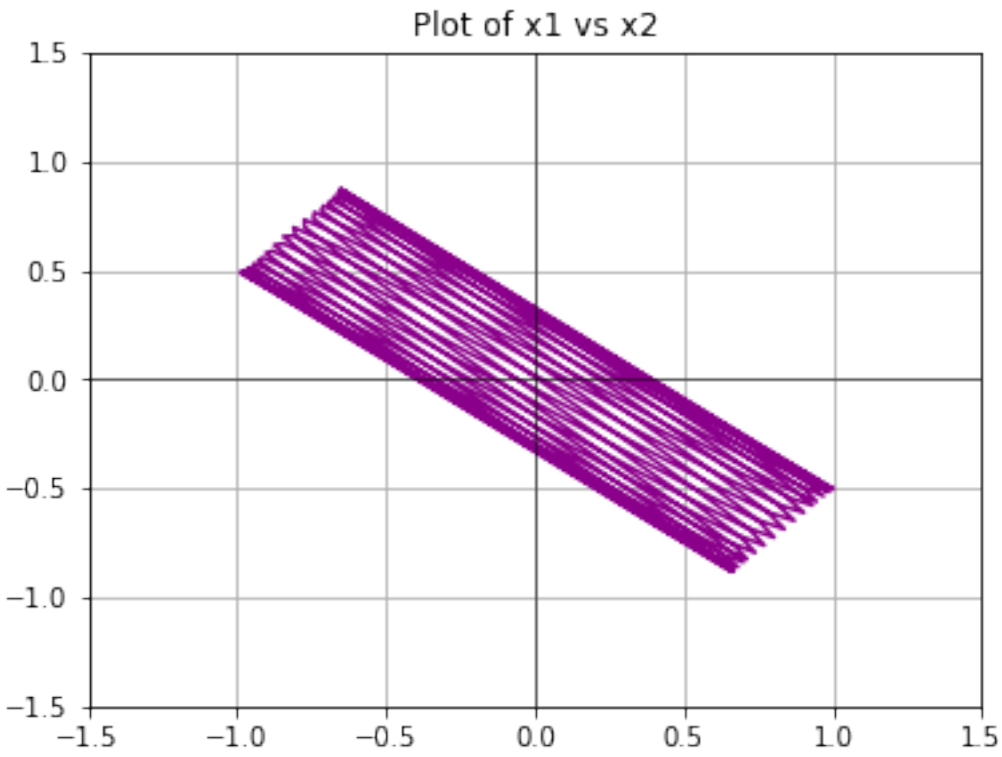

Phase plot for x1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
t
x_1


Phase plot for x2
x_1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
t


Plot of x1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
x_1
0
10
20
30
40
50
t


Plot of x2
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
x_1
0
10
20
30
40
50
t


Plot of x1 and x2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
x_1
x_2
0
10
20
30
40
50
t

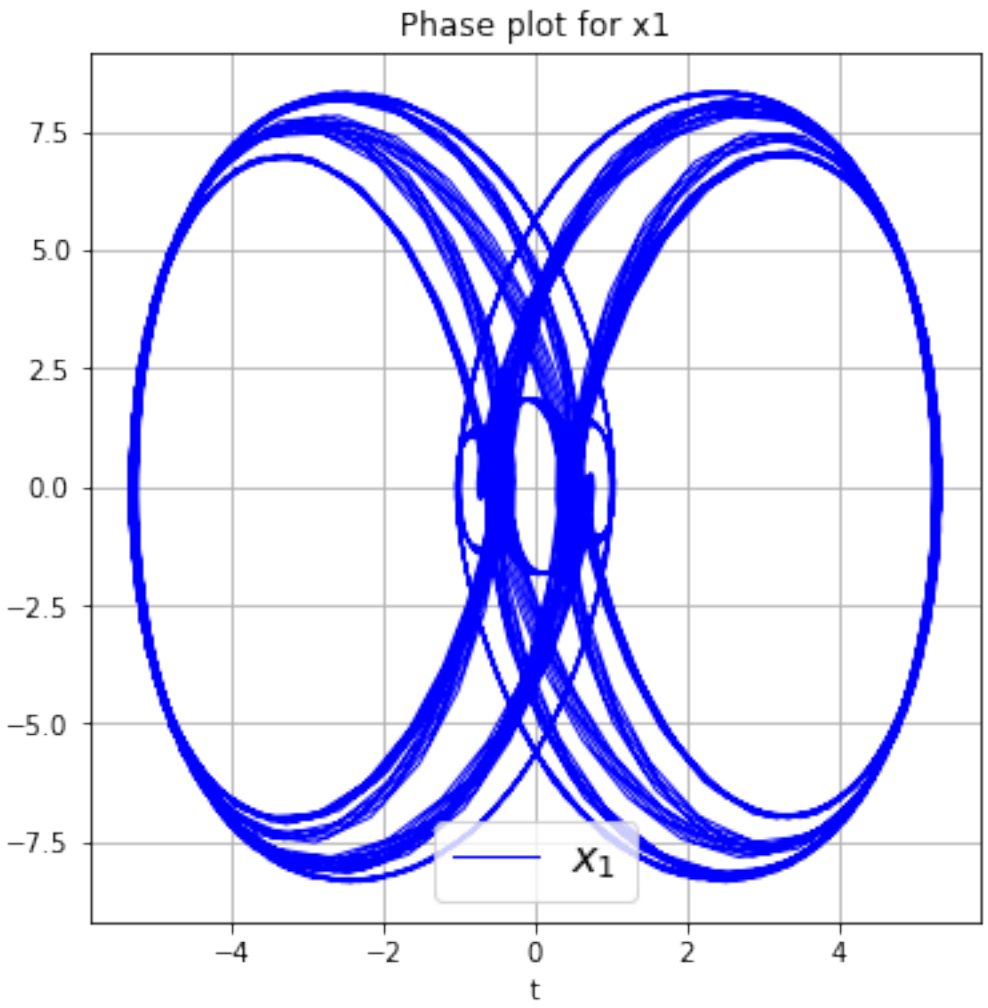


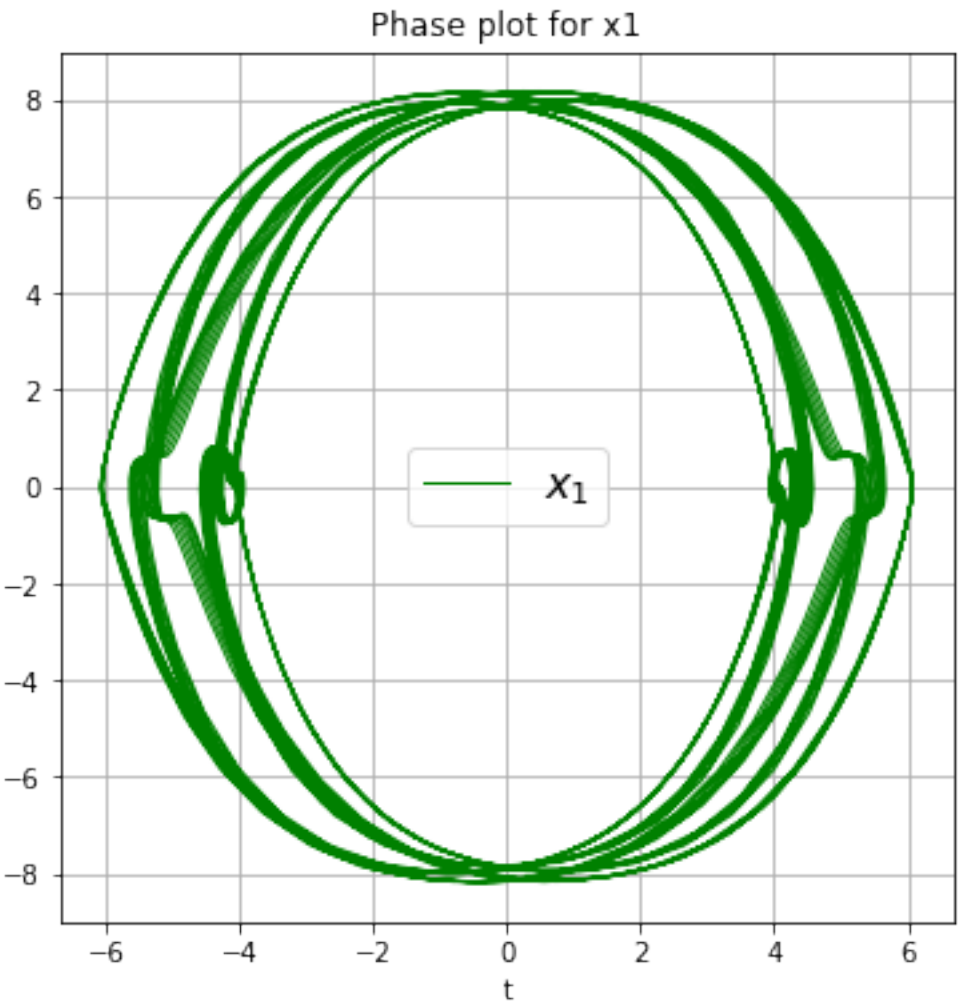
Ejemplo 3.2 Asumiendo $m_1 = m_2 = 1$. Describe el movimiento para un modelo de dos resortes con constantes $k_1 = 0.4$ y $k_2 = 1.808$, coeficientes de amortiguamiento $\delta_1 = 0$ $\delta_2 = 0$, coeficientes no lineales $\mu_1 = -1/6$ y $\mu_1 = -1/10$, con condiciones iniciales $(x_1(0), \dot{x_1}(0), x_2(0), \dot{x_2}(0)) = (-0.5, 1/2, 3.001, 5.9)$. De igual manera solo se modificaron las condiciones iniciales juntos con los parámetros.

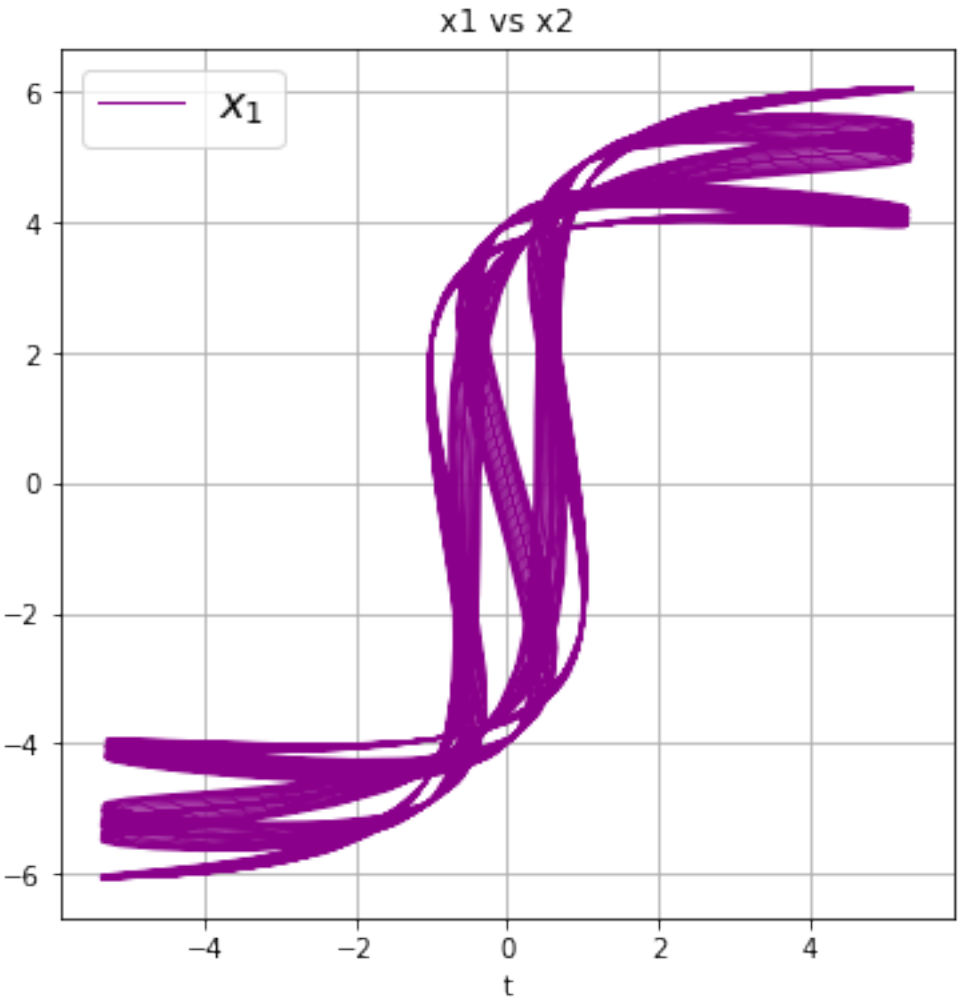
```
In [27]: # Use ODEINT to solve the differential equations defined by the vector field
         from scipy.integrate import odeint

         # 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
```

```
#nonlinearity coefficients
z1 = -(1.0/6.0)
z2 = -(1.0/10.0)

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = -0.5
y1 = 1.0/2.0
x2 = 3.001
y2 = 5.9

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 200.0
numpoints = 2000

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

# Pack up the parameters and initial conditions:
p = [m1, m2, k1, k2, L1, L2, b1, b2, z1, z2 ]
w0 = [x1, y1, x2, y2]

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

with open('two_springsnl2.dat', 'w') as g:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], w1[2], w1[3], file = g)
```

Las gráficas obtenidas son las siguientes:


Phase plot for x1
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−4
−2
0
2
4
t
x_1


Phase plot for x1
8
6
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6
t
x_1

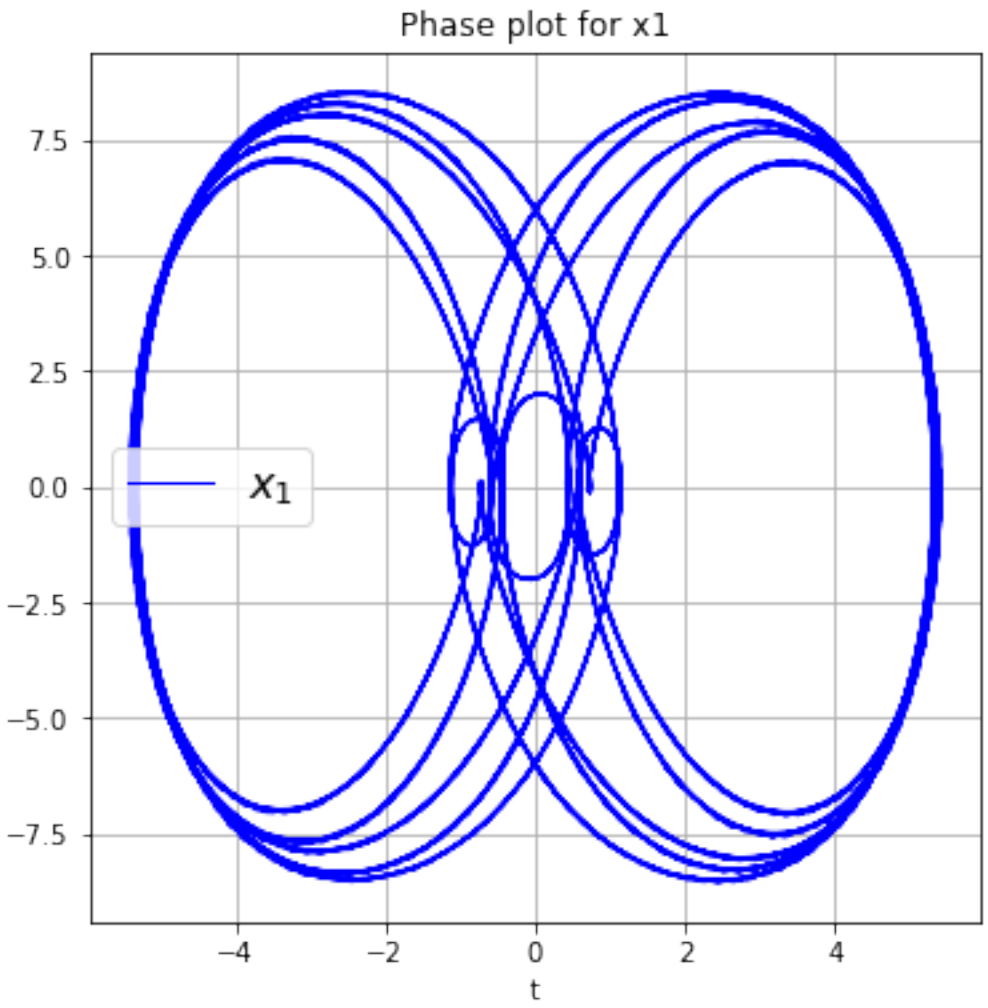


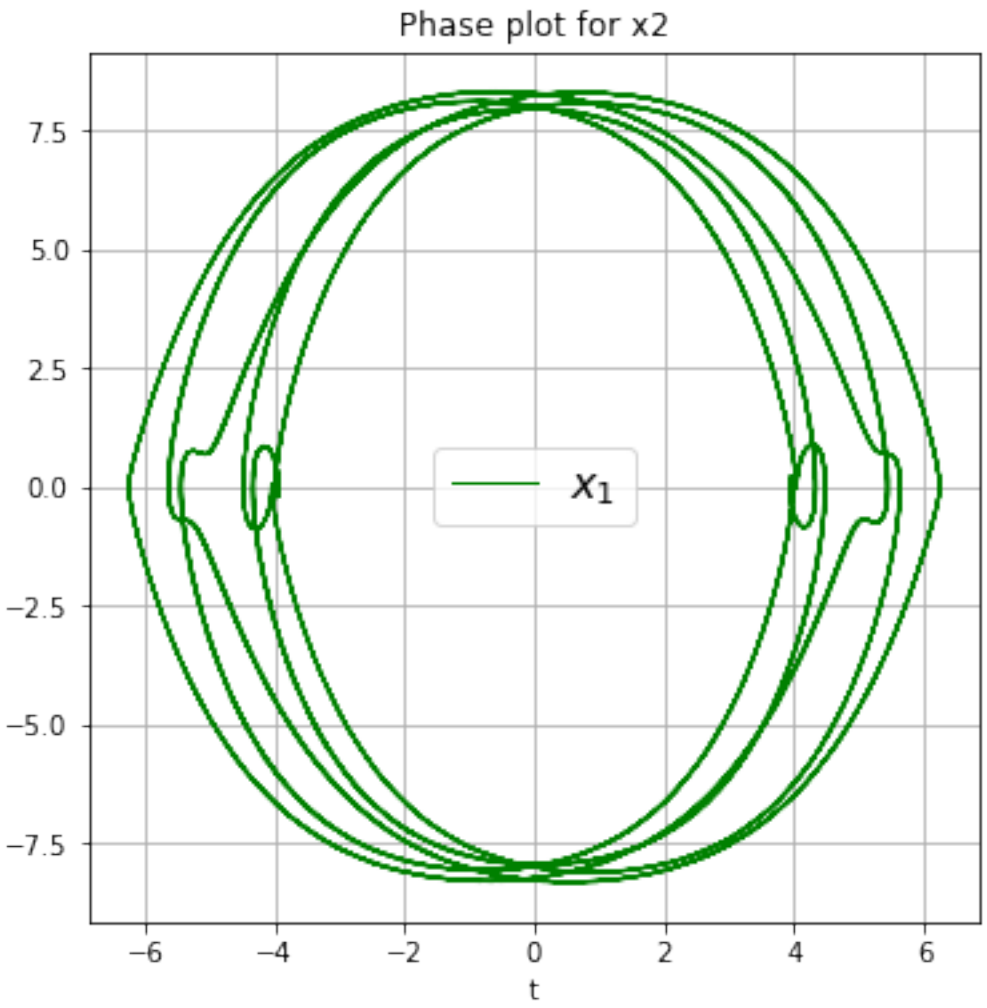
Ejemplo 3.3 Asumiendo $m_1 = m_2 = 1$. Describe el movimiento para un modelo de dos resortes con constantes $k_1 = 0.4$ y $k_2 = 1.808$, coeficientes de amortiguamiento $\delta_1 = 0$ $\delta_2 = 0$, coeficientes no lineales $\mu_1 = -1/6$ y $\mu_1 = -1/10$, con condiciones iniciales $(x_1(0), \dot{x_1}(0), x_2(0), \dot{x_2}(0)) = (-0.6, 1/2, 3.001, 5.9)$. De igual manera solo se modificaron las condiciones iniciales juntos con los parámetros en el segmento de código utilizado.

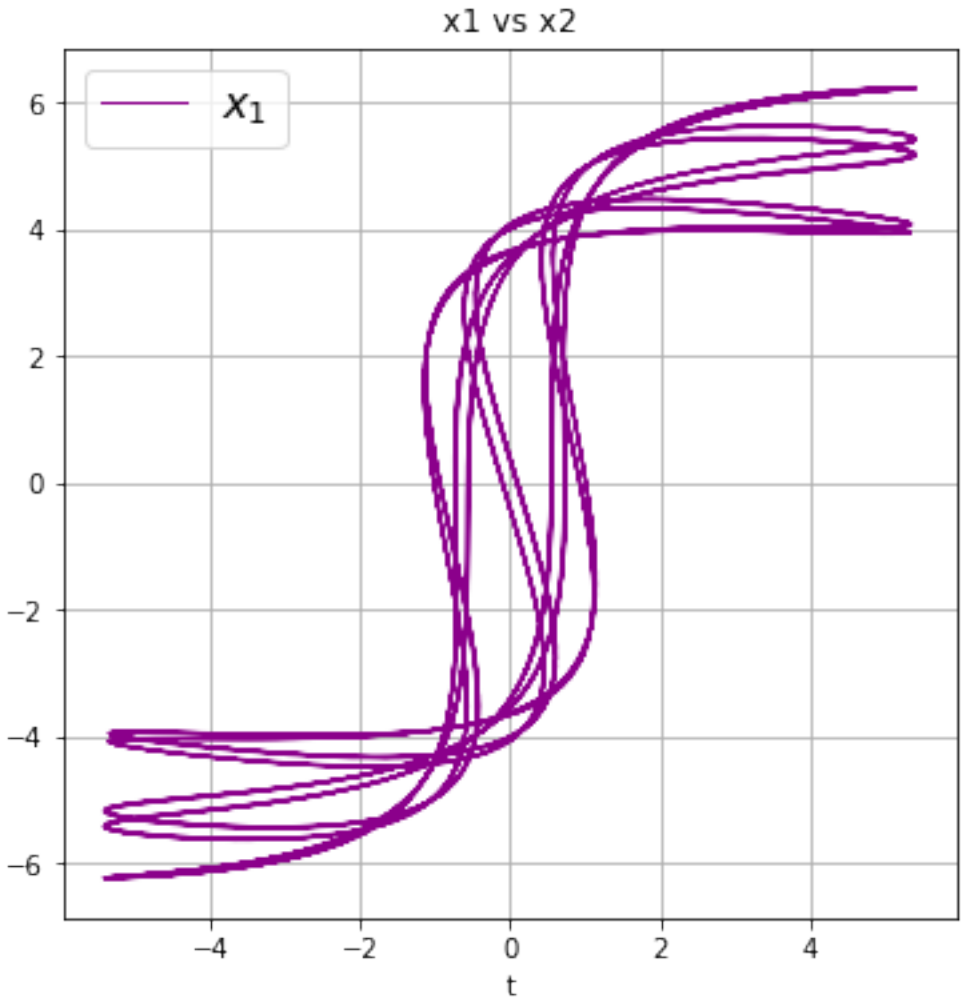
```
In [39]: # Use ODEINT to solve the differential equations defined by the vector field
         from scipy.integrate import odeint

         # 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
#nonlinearity coefficients
z1 = -(1.0/6.0)
z2 = -(1.0/10.0)

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = -0.6
y1 = 1.0/2.0
x2 = 3.001
y2 = 5.9

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 200.0
numpoints = 2000

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

# Pack up the parameters and initial conditions:
p = [m1, m2, k1, k2, L1, L2, b1, b2, z1, z2 ]
w0 = [x1, y1, x2, y2]

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

with open('two_springsnl3.dat', 'w') as h:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], w1[2], w1[3], file = h)
```

Las gráficas obtenidas son las siguientes, donde podemos notar el gran cambio al solo modificar por 0.1 en las condiciones iniciales.


Phase plot for x1
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−4
−2
0
2
4
t
x_1


Phase plot for x2
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−6
−4
−2
0
2
4
6
t
x_1



4. Añadiendo forzamiento

Es una cuestión simple agregar forzamiento externo al modelo. De hecho, podemos manejar cada peso de manera diferente. Supongamos que es un forzamiento sinusoidal simple de la forma $F\cos\omega t$ entonces tenemos

$$m_1\ddot{x}_1 = -\delta_1\dot{x}_1 - k_1x_1 + \mu_1x_1^3 - k_2(x_1 - x_2) + \mu_2(x_1 - x_2)^3 + F_1\cos\omega_1 t$$
$$m_2\ddot{x}_2 = -\delta_2\dot{x}_2 - k_2(x_2 - x_1) + \mu_2(x_2 - x_1)^3 + F_2\cos\omega_2 t$$

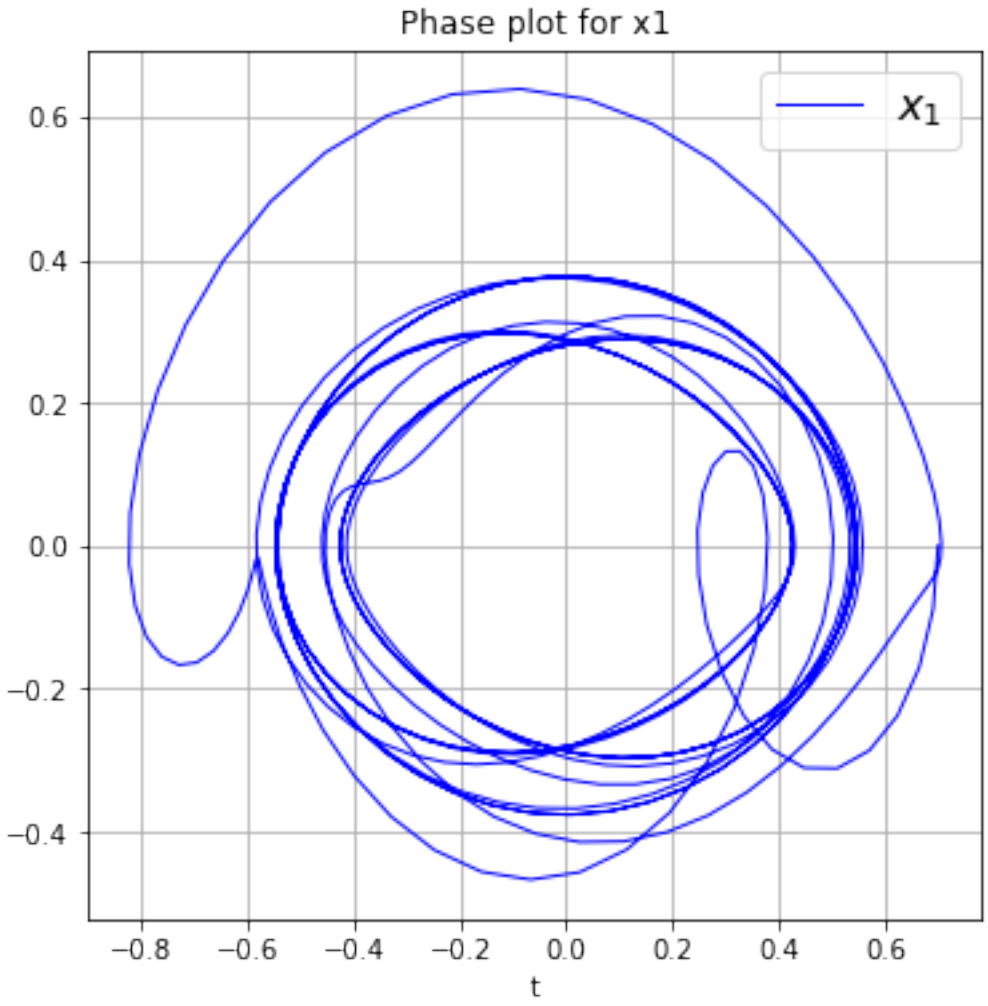
El rango de movimiento para este tipo de modelo tiene muchas posibilidades, resonancia no lineal, el periodo de solución comparte el mismo periodo con el forzamiento e incluso subarmónicas . El código para resolverlo numéricamente vuelve a utilizarse con sus respectivas modificaciones y añadiendo los parámetros correspondientes a esa sección.

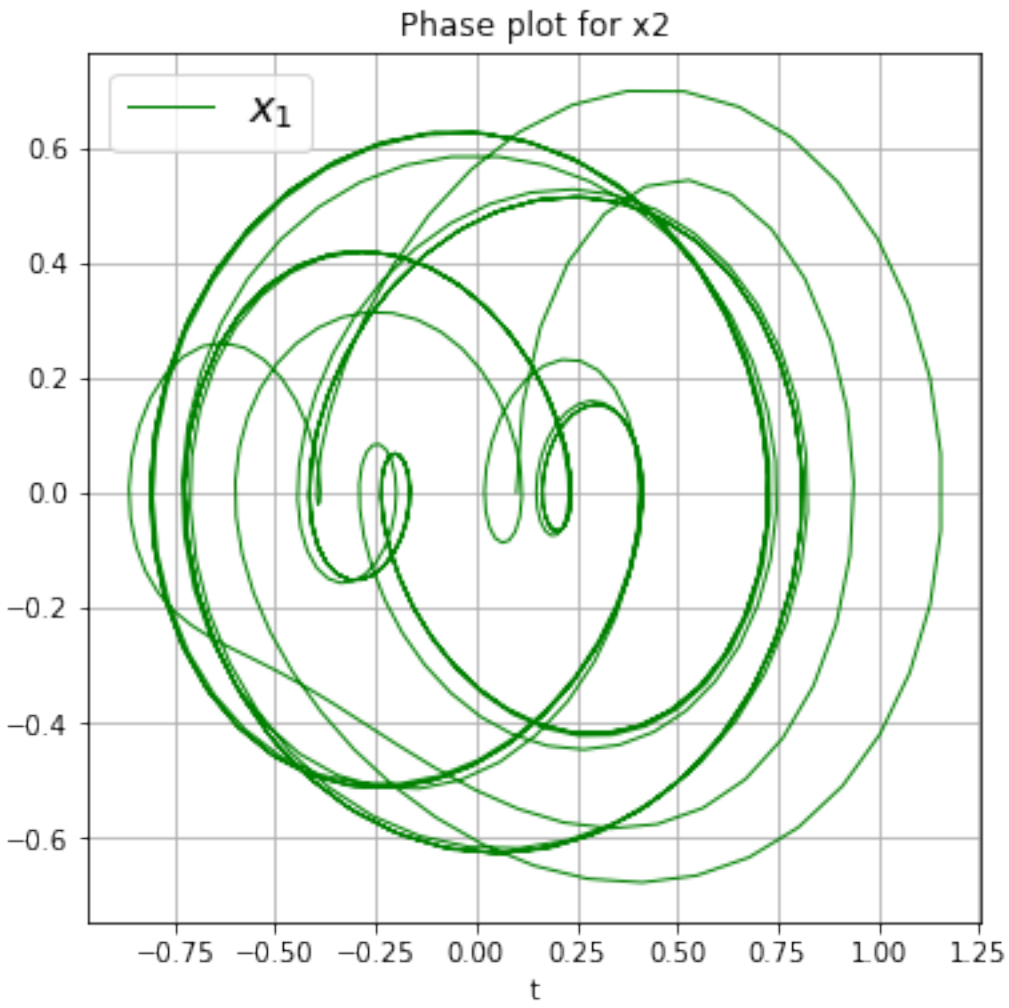
Ejemplo 4.1 Asumiendo $m_1 = m_2 = 1$. Describe el movimiento para un modelo de dos resortes con constantes $k_1 = 2/5$ y $k_2 = 1.$, coeficientes de amortiguamiento $\delta_1 = 1/10$ $\delta_2 = 1/5$, coeficientes no lineales $\mu_1 = 1/6$ y $\mu_1 = 1/10$, amplitud del forzamiento $F_1 = 1/3$ y $F_2 = 1/5$, y frecuencias del forzamiento $\omega_1 = 1$ y $\omega_2 = 3/5$,con condiciones iniciales $(x_1(0), \dot{x_1}(0), x_2(0), \dot{x_2}(0)) = (0.7, 0, 0.1, 0)$.

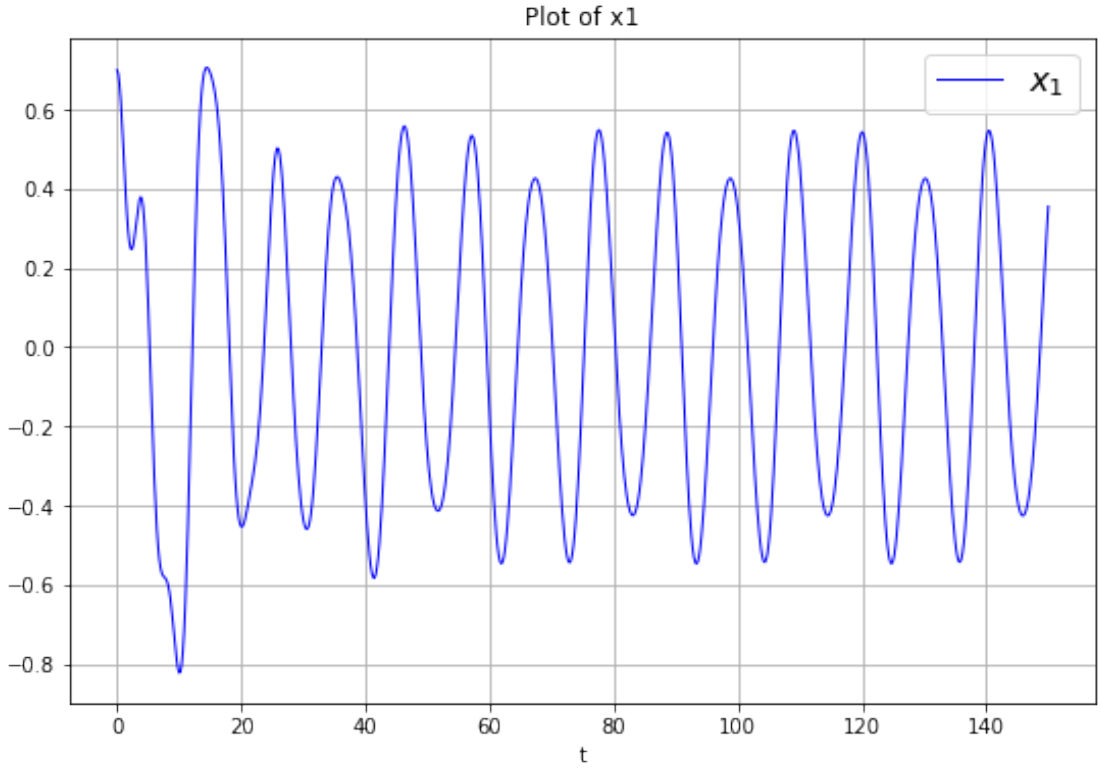
```
In [2]: # Use ODEINT to solve the differential equations defined by the vector field
        from scipy.integrate import odeint

        # Parameter values
        # Masses:
        m1 = 1.0
        m2 = 1.0
        # Spring constants
        k1 = 2.0/5.0
        k2 = 1.0
        # Natural lengths
        L1 = 0.0
        L2 = 0.0
        # Friction coefficients
        b1 = 1.0/10.0
        b2 = 1.0/5.0
        #nonlinearity coefficients
        z1 = 1.0/6.0
        z2 = 1.0/10.0
        #angular frecuency
        u1 = 1
        u2 = 3.0/5.0
        #Forces
        F1 = 1.0/3.0
        F2 = 1.0/5.0
        # 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 = 150.0
        numpoints = 750

        # Create the time samples for the output of the ODE solver.
```

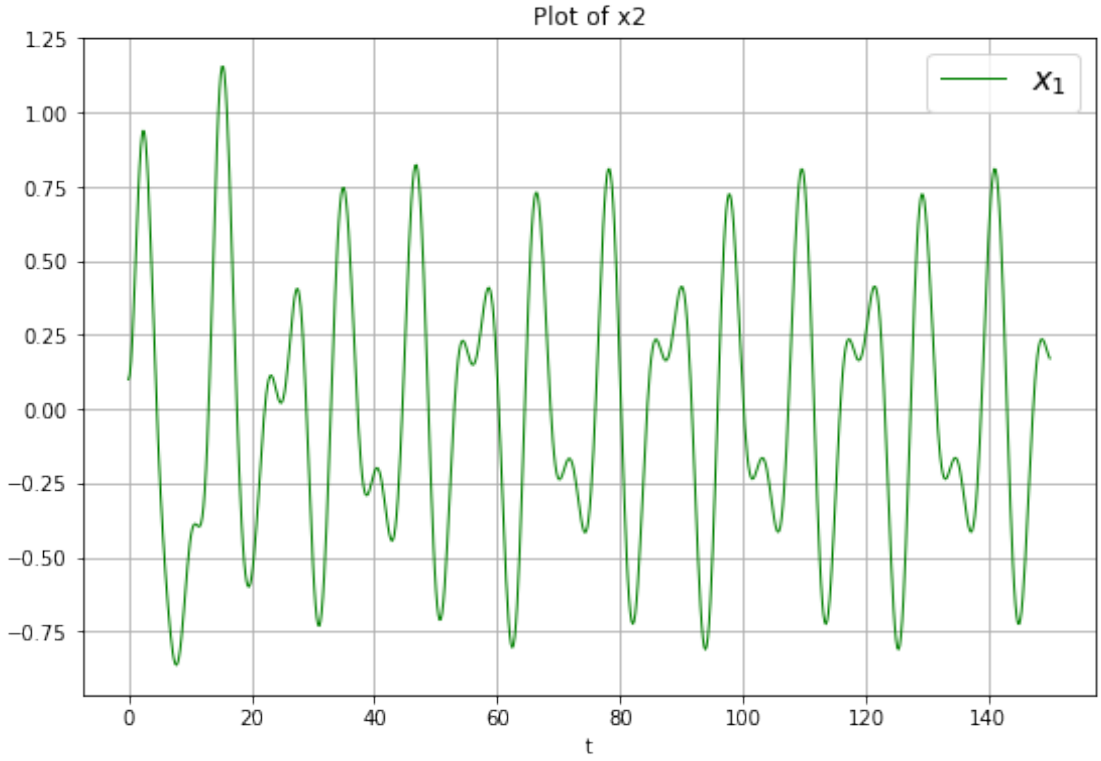
```
# I use a large number of points, only because I want to make
# a plot of the solution that looks nice.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

# Pack up the parameters and initial conditions:
p = [m1, m2, k1, k2, L1, L2, b1, b2, z1, z2, F1, F2, u1, u2 ]
w0 = [x1, y1, x2, y2]

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

with open('two_springsf.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)
```

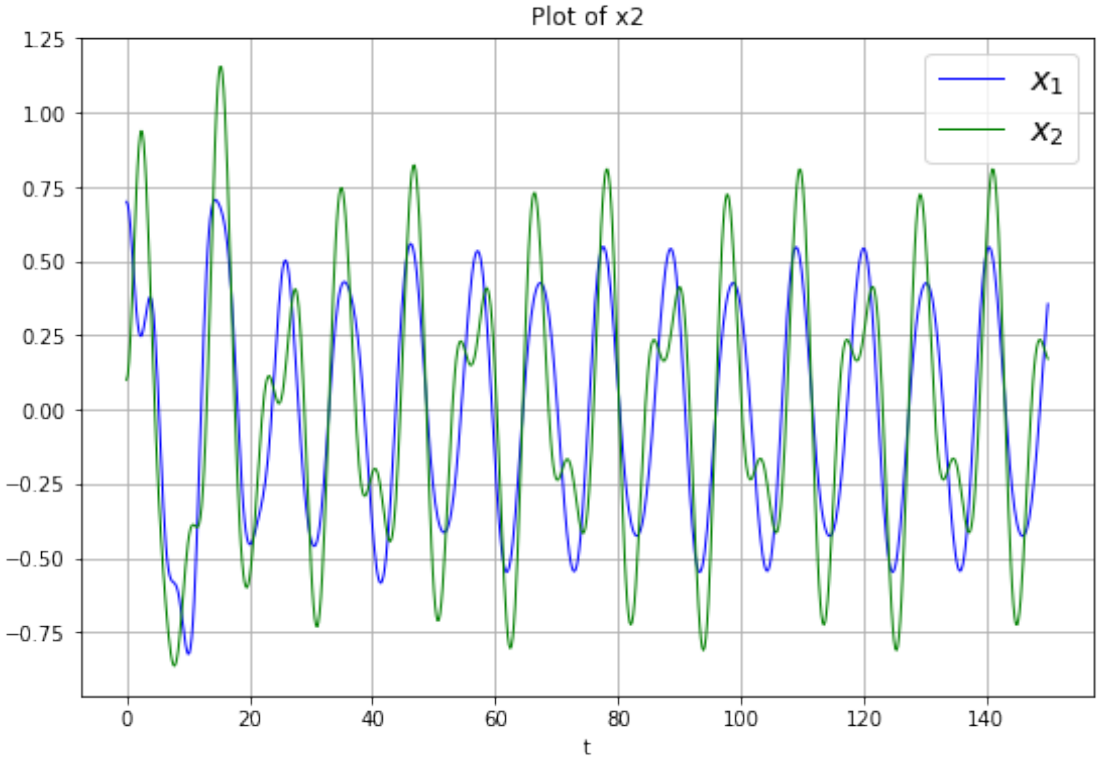
Las gráficas obtenidas son las siguientes:

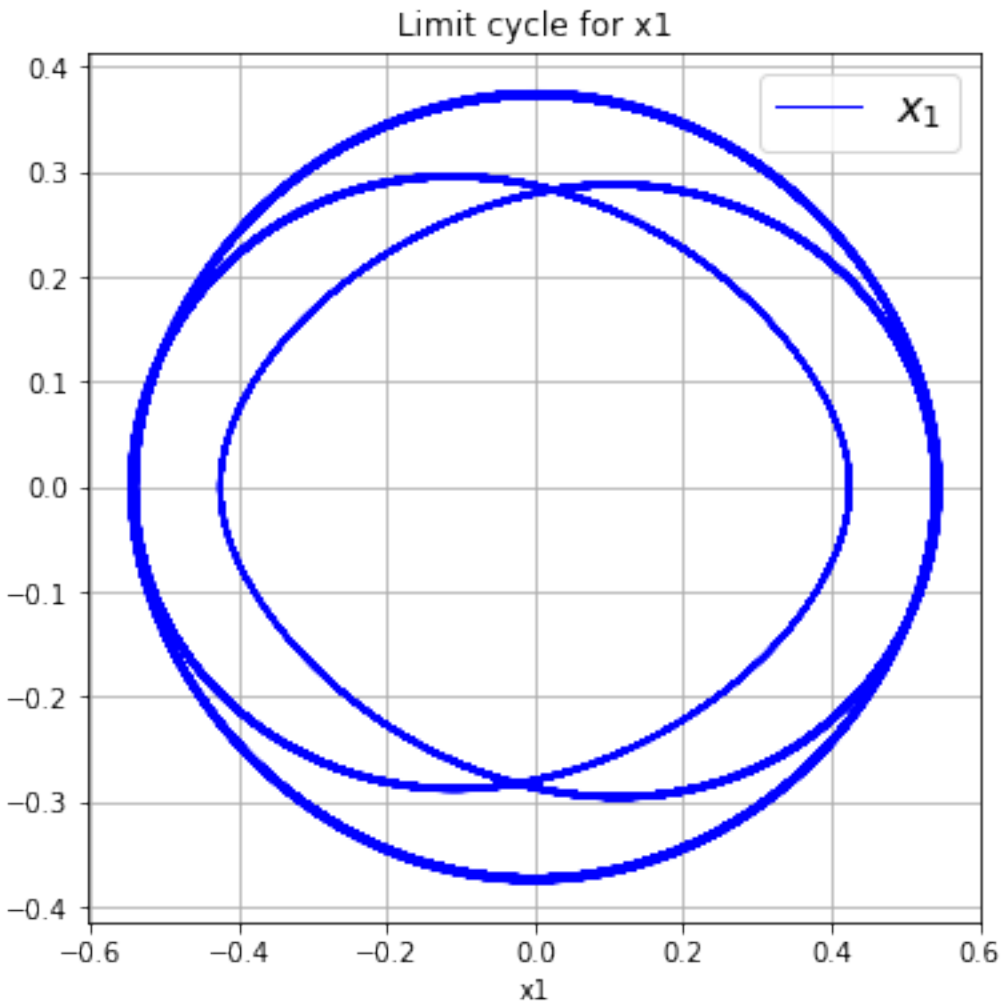


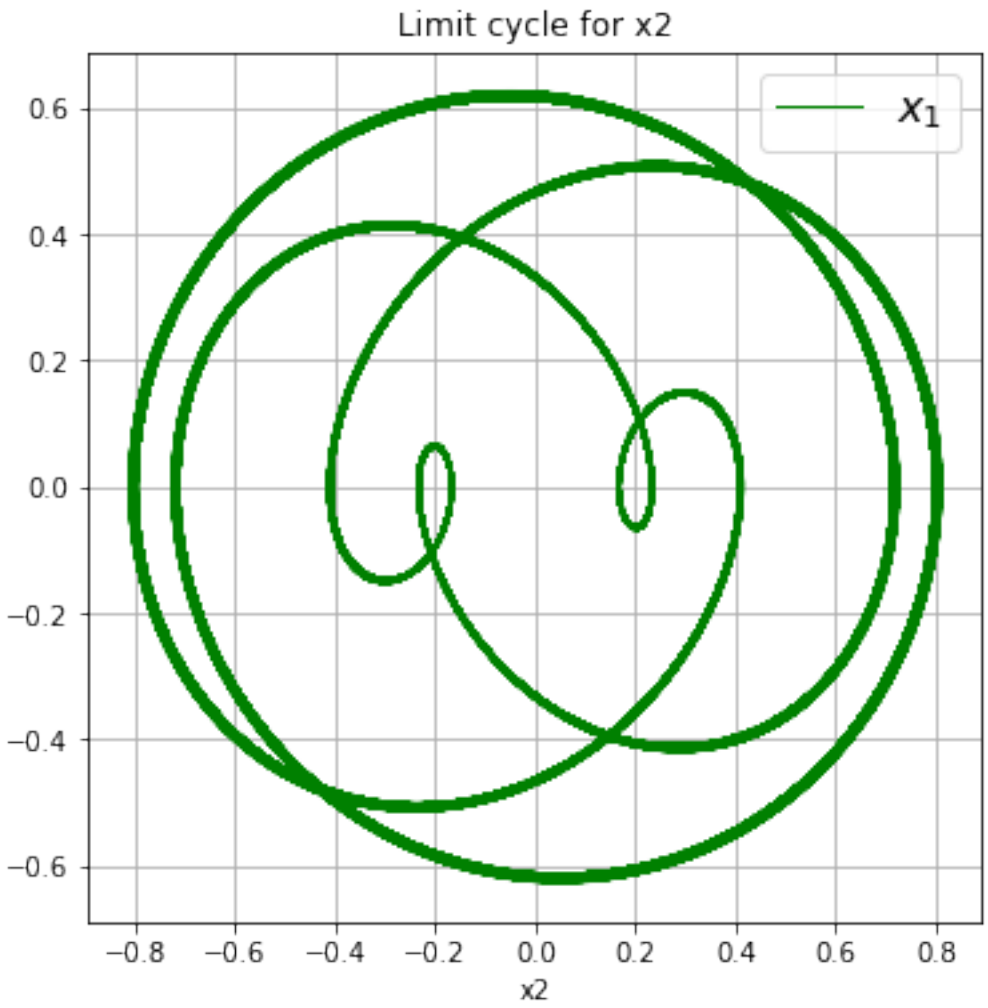

Phase plot for x2
x_1
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
t


Plot of x1
x_1
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
20
40
60
80
100
120
140
t


Plot of x2
x_1
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
20
40
60
80
100
120
140
t


Plot of x2
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
20
40
60
80
100
120
140
t
x_1
x_2


Limit cycle for x1
x_1
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
x1



Conclusiones

Al obtener el total de las soluciones numéricas del documento y comparar las gráficas de las soluciones analíticas con las soluciones numéricas y ver el gran parecido podemos concluir con la afirmación de que los procesos numéricos son de gran ayuda en los momentos de resolución de problemas que se pueden complicar mas de lo normal.

Referencias

[1] Various authors. Coupled spring-mass system. http://scipy-cookbook.readthedocs.io/items/CoupledSpringMassSystem.html, 2009.

[2] The Scipy community. Integration and odes (scipy.integrate). https://docs.scipy.org/doc/scipy/reference/integrate.html, 2017.

[3] FAY TEMPLE H. Coupled spring equations. http://math.oregonstate.edu/~gibsonn/Teaching/MTH323-010S15/Supplements/coupled_spring.pdf, 2002.

Apéndice

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

   - Que podemos apreciar en comparativa todos los ejemplos de problemas que se derivan de las ecuaciones de un sistema de resortes acoplados ya que por fin esta toda la información junta en un solo texto.

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?

   - Puede ser un punto de partida para analizar nuevos fenómenos no lineales ya que esto se pudiera encontrar en mayor proporción en la naturaleza que casos idealmente lineales.

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

   - Propondría una clase teórica superficial donde al menos se resuelvan las ecuaciones para encontrar de manera analítica la solución y así poder entender de mejor forma la dinámica de estas ecuaciones y los problemas a analizar que conllevan.

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

   - Seria interesante.