Reporte - Actividad 7: Sistema de Masas con Oscilaciones No Lineales y Forzadas

García Monge Itzel Alexia

24 de Marzo, 2018

1 Introducción

El siguiente reporte se centra en encontrar la solución de un sistema de dos resortes acoplados, contando con oscilaciones lineales, no lineales y forzamiento, siendo su solución un sistema de ecuaciones lineales de cuarto orden. Se utilizó la herramienta *JupyterLab* para programar con *Python* y lograr conseguir el valor de las ecuaciones recurriendo a métodos numéricos y graficando el resultado, seleccionando un cierto número de puntos para afinar el valor obtenido. Se comparó la solución analítica conocida y el valor aproximado de *Python*, obteniendo su error y, nuevamente, graficarlo.

2 Síntesis

2.1 Introducción

En los últimos años, la elaboración de programas que resuelven sistemas algebraicos con algoritmos numéricos de gran potencia y con la capacidad de crear gráficas fácilmente ha ayudado a crear un nuevo estándar en la búsqueda de soluciones de ecuaciones lineales, especialmente creando un enfoque en las soluciones no lineales.

Este artículo se centra en el clásico ejemplo de dos resortes con pesos atados en serie colgando a una altura específica. Asumiendo que se comportan acorde a las leyes de Hooke podemos modelar el problemas con un par de ecuaciones diferenciales lineales de segundo orden. Al diferenciar y sustituir una ecuación de la otra obtenemos que el movimiento de cada masa puede determinarse por una ecuación diferencial lineal de cuarto orden.

En una ecuación diferencial de cuarto orden podemos investigar cuándo los movimientos de las dos masas se encuentran en fase o desfase. Si variamos la constante de resorte o agregamos un factor–aunque sea mínimo–no lineal, los movimientos oscilatorios producidos actúan de una manera muy distinta a la del modelo clásico de un resorte.

2.2 Sistema de dos Resortes

El modelo de dos resortes y dos masas consiste en tener un resorte con una constante de k_1 atado a un techo con una masa m_1 colgando de ella. A esta masa se le agrega un segundo resorte con constante k_2, colgando una masa m_2 bajo todo el sistema. Cuando el sistema se encuentra en equilibrio, medimos la distancia del centro de masa m_1 y m_2 hasta el equilibrio. Denominamos a las distancias x_1 y x_2, obteniendo un modelo como el que se observa en la figura siguiente.

2.2.1 Asumiendo la Ley de Hooke

Si asumimos que existen pequeñas oscilaciones, las fuerzas restauradoras serían de la forma $-k_1 l_1$ y $-k_2 l_2$ donde l_1 y l_2 son la elongación o compresión de los resortes.

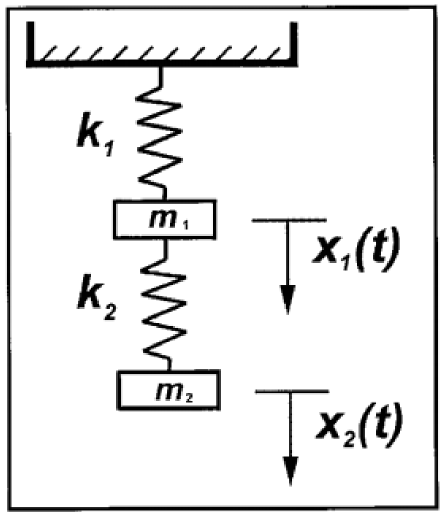


Figure 1: Dos resortes acoplados

Al estar m_1 conectada a los dos resortes, se tienen dos fuerzas restauradoras actuando sobre ella. Una fuerza ascendente $-k_1x_1$ ejercida por x_1 del primer resorte; otra fuerza ascendente $-k_2(x_2-x_1)$ ejercida de la resistencia del segundo resorte a ser comprimido o elongado por x_2-x_1. Mientras que m_2 solo es afectada por la fuerza restauradora del segundo resorte. Al asumir que no hay fricción y usando las Leyes de Newton obtenemos un sistema de ecuaciones lineales de segundo orden de la forma

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

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

2.2.2 Algunos Ejemplos con Masas Idénticas

Para encontrar la ecuación de x_1 que no involucré a x_2, resolvemos la primera ecuación de x_2, despejándola para después sustituirla en la segunda ecuación diferencial. Simplificando la ecuación obtenemos

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

Repetimos el mismo procedimiento para encontrar x_2 y obtenemos la ecuación

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

En ambos casos se tiene una ecuación diferencial de cuarto orden muy parecidas entre si en el movimiento de la primera masa. Es decir, los movimientos de cada masa obedecen la misma ecuación diferencial, y son solo las posiciones y velocidades iniciales las necesarias para determinar cualquier caso específico.

Si consideramos que $m_1 = m_2$, asumiendo que no hay fricción ni fuerzas externas, nuestra ecuación diferencial se reduce a

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

Ejemplo 2.1 *Describe el movimiento para resortes constantes $k_1 = 6$ y $k_2 = 4$ con una condición inicial (x_1(0), $\dot{x}_1$(0), x_2(0), $\dot{x}_2$(0))=(1,0,2,0)*

Resolviendo de manera analítica, obtenemos una solución única para x_1 y x_2 que dependen de t:

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

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

El movimiento que se tiene es uno sincronizado, es decir, los sistemas están en fase–se tiene el mismo periodo de movimiento pero se cuentan con diferentes amplitudes. Como el movimiento en una movimiento periódico simple, las fases de x_1 y x_2 forman elipses.

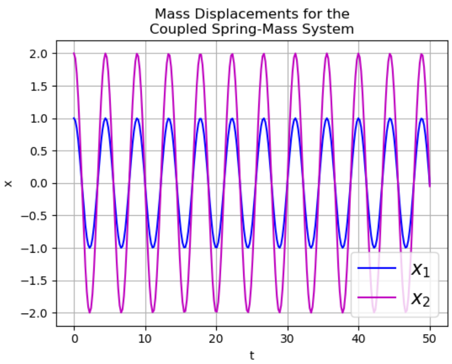
Al contar con los valores de ambas masas y constantes de resorte tenemos lo necesario para graficar el movimiento utilizando *Python*. Los demás ejemplos se graficarán de la misma forma, únicamente variando los valores de m, k y b proporcionados en el segundo segmento de código.

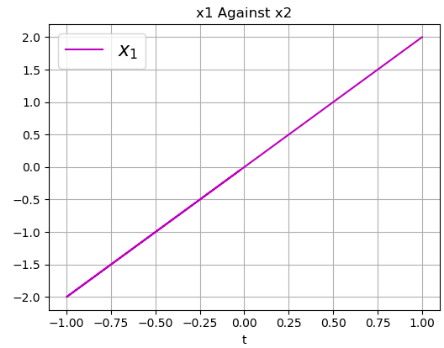
```
def vectorfield(w, t, p):
    """
    Definimos las ecuaciones diferenciales para el sistema de doble masa-resorte.
    Arguments:
        w : Vector del estado de las variables
                  w = [x1,y1,x2,y2]
        t :  Tiempo
        p :  Vector de los parametros:
                  p = [m1,m2,k1,k2,L1,L2,b1,b2]
    """
    x1, y1, x2, y2 = w
    m1, m2, k1, k2, L1, L2, b1, b2 = p

    #Creamos 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
```

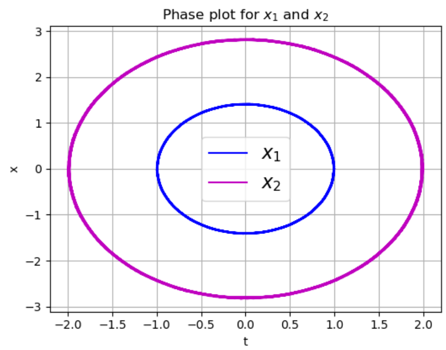
```python
from scipy.integrate import odeint
import numpy as np

#Valor de los parametros
# Masas:
m1 = 1.0
m2 = 1.0
# Constante del resorte
k1 = 6.0
k2 = 4.0
# Longitudes naturales
L1 = 0
L2 = 0
# Coeficientes de fricción
b1 = 0.0
b2 = 0.0

# Condiciones iniciales
# x1 and x2 son las pocisiones iniciales(contando la longitud de L), y y1 y y2 son las velocidades
x1 = 1.0
y1 = 0.0
x2 = 2.0
y2 = 0.0

# Parametros de la ED
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50.0
numpoints = 250

#Creamos los valores del tiempo, entre mas puntos damos, mejores se veran las graficas.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

# Ponemos a las variables en un vector.
p = [m1, m2, k1, k2, L1, L2, b1, b2]
w0 = [x1, y1, x2, y2]

# Llamamos a la funcion para resolver la ED
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

with open('ejemplo2.1.dat', 'w') as f:
    # Imprimimos en el documento la solución
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], w1[2], w1[3],np.abs((w1[0]-(np.cos(np.sqrt(2.0)*t1)))/(np.cos(np.sqrt(2.0)*t1))),
               np.abs((w1[2]-(2.0*np.cos(np.sqrt(2.0)*t1)))/(2.0*np.cos(np.sqrt(2)*t1))),file=f)
```

```python
#Graficamos la solución

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

t, x1, xy, x2, y2, er1, er2 = loadtxt('ejemplo2.1.dat', unpack=True)

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

xlabel('t')
ylabel('x')
grid(True)
lw = 1.5

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

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Mass Displacements for the\nCoupled Spring-Mass System')
savefig('Ejemplo2.1_pt1.png', dpi=100)
```

Obteniendo gráficas de la forma:







Ejemplo2.2 *Describe el movimiento para la constante de los resortes k_1=6, k_2=4 con condiciones iniciales (x_1(0), $\dot{x}_1$(0), x_2(0), $\dot{x}_2$(0))=(-2,0,1,0)*

Resolvemos nuevamente de manera analítica y obtenemos una solución única para x_1 y x_2 que dependen de t:

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

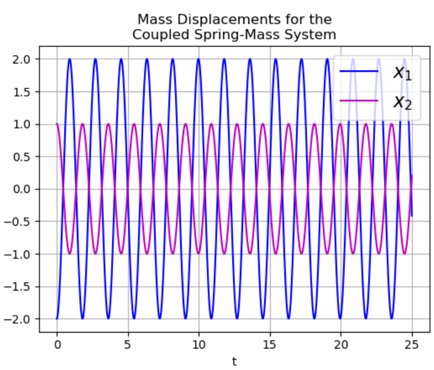
$$x_2(\mathrm{t})=\cos 2\sqrt{3}\mathrm{t}$$

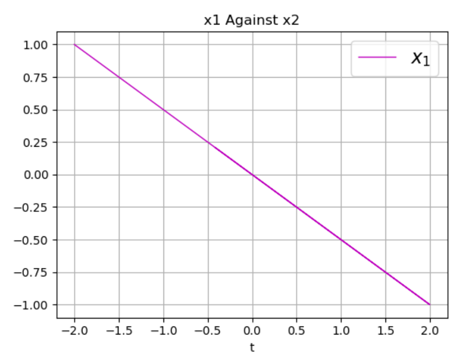
Cuando m_1 se mueve de manera descendente, m_2 lo hace de manera ascendente y vice-versa. Nuevamente ambos movimientos tienen el mismo periodo, pero ahora se encuentran 180° fuera de fase.

Para graficar los valores encontrados modificamos el segundo segmento de código mostrado en el ejemplo anterior

```python
from scipy.integrate import odeint
import numpy as np

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 6.0
k2 = 4.0
# Natural lengths
L1 = 0.0
L2 = 0.0
# Friction coefficients
b1 = 0.0
b2 = 0.0

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

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 25.0
numpoints = 1250

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

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

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

with open('ejemplo2.2.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]-(-2.0*np.cos(2.0*np.sqrt(3.0)*t1)))/(-2.0*np.cos(2.0*(np.sqrt(3.0)*t1)))),
              np.abs((w1[2]-(np.cos(2.0*np.sqrt(3.0)*t1)))/(np.cos(2.0*np.sqrt(3.0)*t1))), file=f)
```

Se terminan obteniendo las siguientes gráficas:





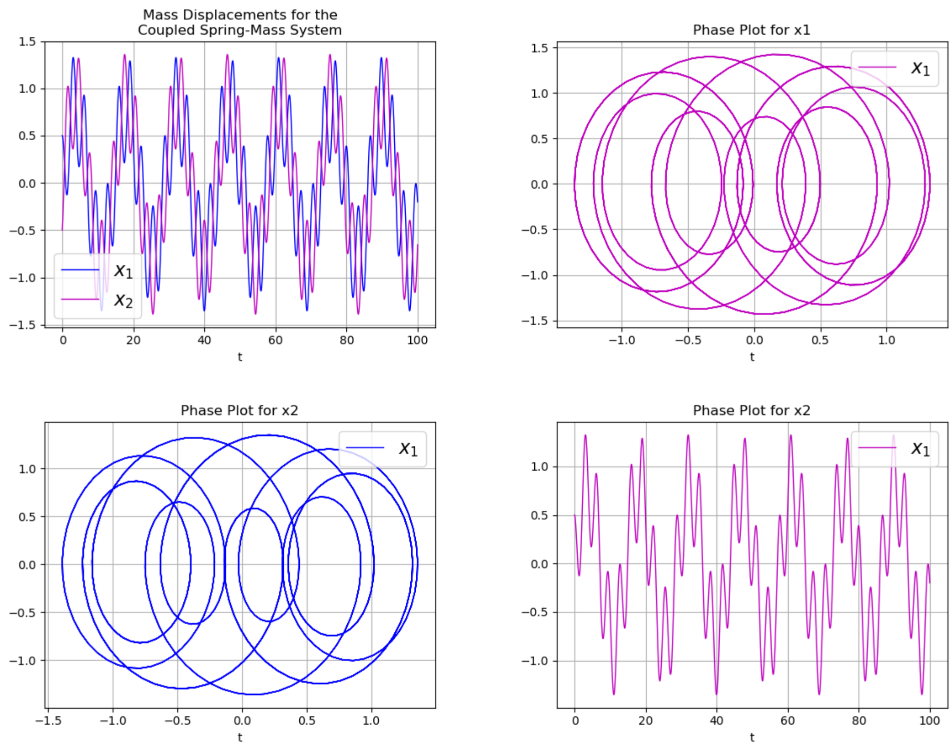
Ejemplo 2.3 *Describe el movimiento para la constante de los resortes* k_1*=0.4,* k_2*=1.808 con condiciones iniciales* $(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0))$*=(1/2,0,-1/2,7/10)*

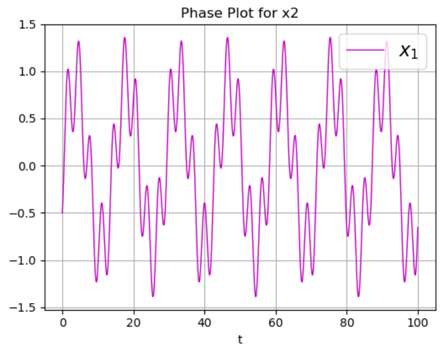
En este ejemplo se muestra como las condiciones iniciales afectan solamente a la amplitud y fase de las soluciones. Son los valores otorgados en las constantes de resorte k_1 y k_2 las que determinan el periodo y, por ende, la frecuencia de respuesta. Sus gráficas de fase tienen casi un movimiento periódico entre ellos, y al graficar x_1 contra x_2 obtenemos una curva tipo Lissajous.

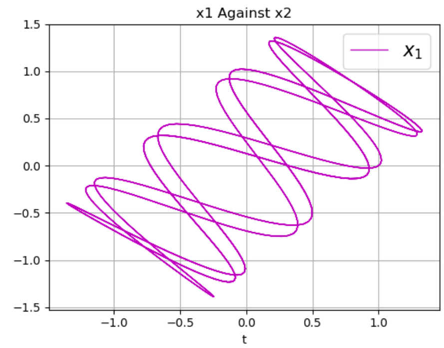
Aunque no se cuente con una respuesta analítica, aún se cuentan con las condiciones iniciales para el código.

```python
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
#Si quiero escribir fracciones, debo de hacerlo como reales y no como enteros, es decir:
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 = 100.0
numpoints = 1250

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

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

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

with open('ejemplo2.3.dat', 'w') as f:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print(t1, w1[0], w1[1], w1[2], w1[3], file=f)
```

Las gráficas resultantes son







2.2.3 Amortiguamiento

El problema de amortiguamiento más encontrado es el de amortiguamiento viscoso, en donde el amortiguamiento es proporcional a su velocidad. El amortiguamiento en m_1 depende solamente en su velocidad, no en la velocidad de m_2 y vice-versa.

Agregamos el amortiguamiento viscoso como δ al modelo que hemos construido asumiendo que δ_1 y δ_2 son valores pequeños. Esto transforma nuestras ecuaciones iniciales en

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

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

Volvemos a realizar el procedimiento inicial, despejando x_2 de la primera ecuación y sustituyéndola en la segunda para conseguir una ecuación con x_1 que no dependa de x_2 y repetimos el proceso con x_1. El resultado que obtenemos es la misma ecuación diferencial lineal representando el movimiento de dos masas, consiguiendo movimientos de oscilaciones amortiguadas bajo simples suposiciones en los parámetros.

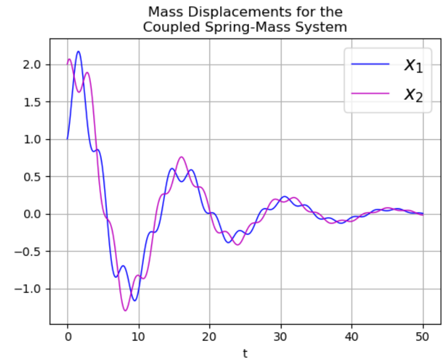
Ejemplo 2.4 *Asume que m_1=m_2=1. Describe el movimiento para la constante de los resortes k_1=0.4, k_2=1.808, coeficientes de amortiguamiento δ_1=0.1 y δ_2 con condiciones iniciales (x_1(0), $\dot{x}_1$(0), x_2(0), $\dot{x}_2$(0))=(1,1/2,2,1/2)*

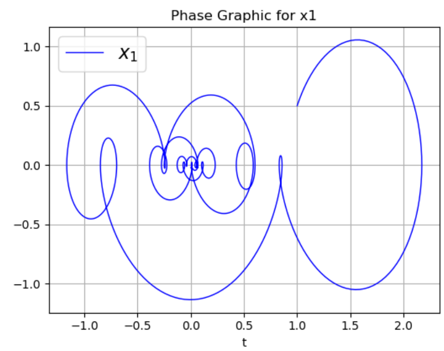
En el retrato de fase para x_1 y x_2 se puede observar un patrón de movimiento con amplitud decreciendo. El movimiento del amortiguamiento oscilatorio es evidente al graficar x_1 y x_2, mientras que al graficar ambas posiciones contra el tiempo encontramos movimientos casi sincronizados, por último, x_1 contra x_2 encontramos un movimiento de oscilación amortiguada de ambas masas.

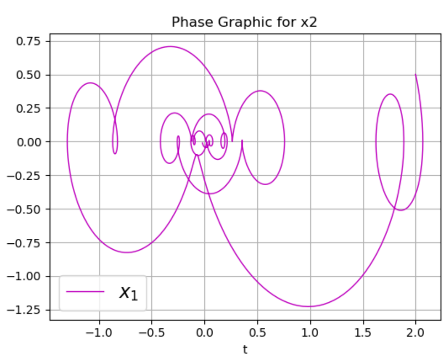
Los cambios que sufrió el segundo segmento de código para graficar estos casos se aprecian a continuación.

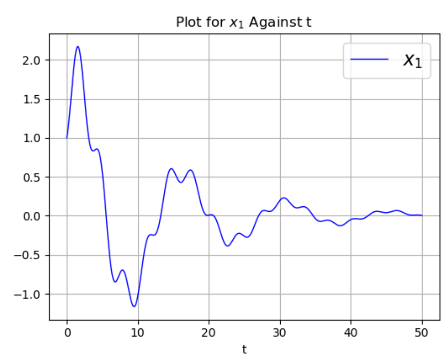
```python
from scipy.integrate import odeint

# 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
#Si quiero escribir fracciones, debo de hacerlo como reales y no como enteros, es decir:
x1 = 1.0
y1 = 1.0/2.0
x2 = 2
y2 = 1.0/2.0

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

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

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

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

with open('ejemplo2.4.dat', 'w') as f:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print(t1, w1[0], w1[1], w1[2], w1[3], file=f)
```

Las gráficas resultantes fueron:









2.3 Añadiendo No Linealidad

Si asumimos que las fuerzas restauradoras son no lineales, estas ya no obedecen la Ley de Hooke y nuestra fuerza restauradora deja de ser $-kx$. En cambio, se obtiene una nueva fuerza restauradora, creando un nuevo modelo de ecuaciones de la forma:

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

El rango de movimiento para los modelos no lineales se vuelve más complicado que los modelos lineales, siendo la precisión un factor importante al solucionar este tipo de ecuaciones. No hay ningún solucionador no numérico que se pueda esperar mantener preciso después de largos intervalos de tiempo.

Ejemplo 3.1 *Asume m_1=m_2=1. Describe el movimiento para las constantes de resorte k_1=0.4 y k_2=1.808, coeficientes de amortiguamiento δ_1=0 y δ_2=0, coeficientes no lineales μ_1=-1/6 y μ_2=-1/10, con condiciones iniciales (x_1(0), $\dot{x}_1$(0), x_2(0), $\dot{x}_2$(0))=(1,0,-1/2,0)*

Al no tener amortiguamiento, los movimientos son oscilatorios y parecen ser periódicos. Además, debido a la no linealidad, el modelo puede mostrar sensibilidad a las condiciones iniciales.

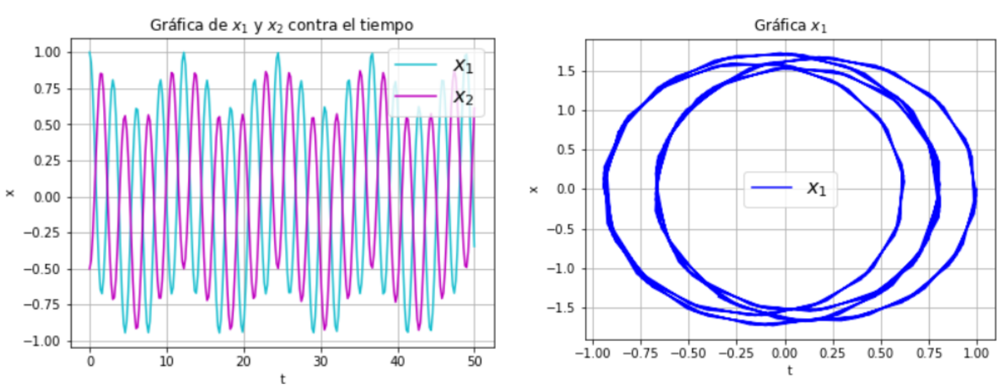
Para lograr graficar ecuaciones no lineales, no basta con modificar el segundo segmento de código como comúnmente lo hicimos en los ejemplos anteriores, pues la ecuación ya no es la misma. Tomando en consideración la no linealidad, las primeras celdas de código se vuelven de la forma siguiente.

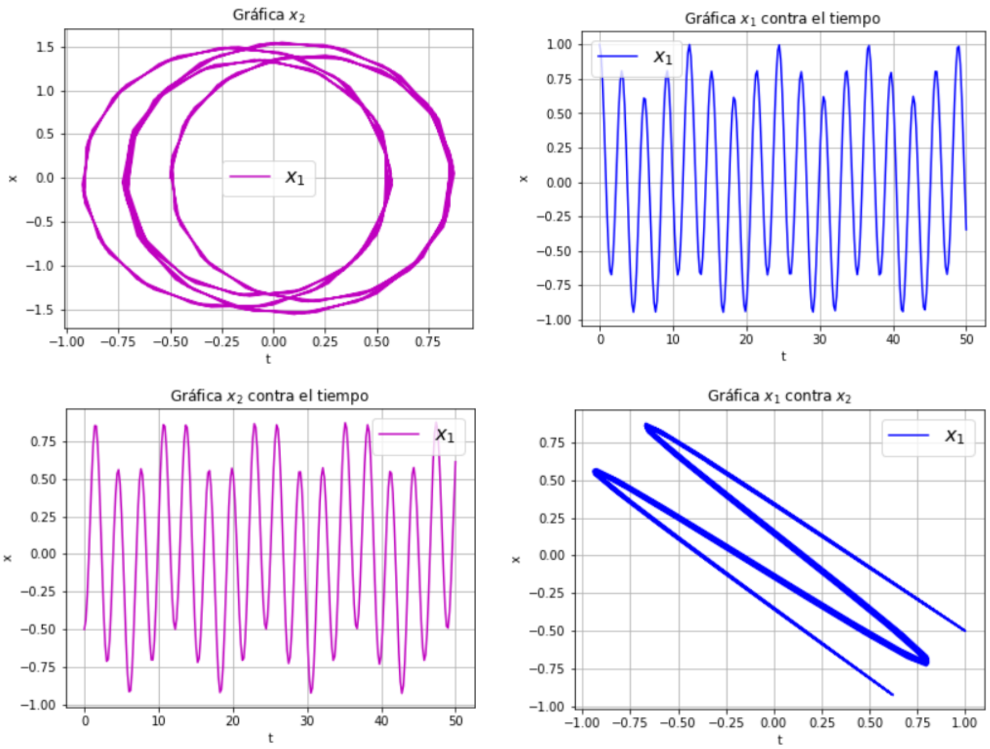
```
#Ejemplo 3.1
def vectorfield(w, t, p):
    """
    Definimos las ecuaciones diferenciales para el sistema de doble masa-resorte.
    Arguments:
        w : Vector del estado de las variables
                  w = [x1,y1,x2,y2]
        t :  Tiempo
        p :  Vector de los parametros:
                  p = [m1,m2,k1,k2,L1,L2,b1,b2]
    """
    x1, y1, x2, y2 = w
    #Agregamos los coeficientes no lineales al vector de parámetros
    m1, m2, k1, k2, L1, L2, b1, b2, c1, c2 = p

    #Creamos f = (x1',y1',x2',y2')
    #Como vamos a usar una fuerza no lineal, debemos modificar la ecuación que trabajamos en la Actividad
 pasada
    f = [y1,
         (-b1 * y1 - k1 * (x1 - L1) + c1 * (x1 - L1) ** 3 - k2 * (x1 - x2 - L2) + c2 * (x1 - x2- L2) ** 3)
 / m1,
         y2,
         (-b2 * y2 - k2 * (x2 - x1 - L2) + c2 * (x2 - x1 - L2) ** 3) / m2]
    return f
```

```python
#Usamos la funcion ODEINT para resolver las ecuaciones diferenciales definidas por el vector
from scipy.integrate import odeint
import numpy as np

#Valor de los parametros
# Masas:
m1 = 1.0
m2 = 1.0
# Constante del resorte
k1 = 0.40
k2 = 1.808
# Longitudes naturales
L1 = 0
L2 = 0
# Coeficientes de fricción
b1 = 0.0
b2 = 0.0

#Parametros de coeficientes no lineales
c1=-1.0/6.0
c2=-1.0/10.0

# Condiciones iniciales
# x1 and x2 son las pocisiones iniciales(contando la longitud de L), y y1 y y2 son las velocidades
x1 = 1.0
y1 = 0.0
x2 = -1.0/2.0
y2 = 0.0

# Parametros de la ED
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50.0
numpoints = 250

#Creamos los valores del tiempo, entre mas puntos damos, mejores se veran las graficas.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

# Ponemos a las variables en un vector.
#Agregamos los coeficientes no lineales
p = [m1, m2, k1, k2, L1, L2, b1, b2, c1, c2]
w0 = [x1, y1, x2, y2]

# Llamamos a la funcion para resolver la ED
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

with open('ejemplo3.1.dat', 'w') as f:
    # Imprimimos en el documento la solución
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], w1[2], w1[3], file=f)
```

La tercera celda no recibe ninguna modificación, siendo la primera y la segunda necesarias al presentarse un cambio en el sistema de ecuaciones en donde se añadieron nuevos factores. Para los siguientes ejemplos nuevamente se mantienen todas las celdas iguales a excepción de la segunda.

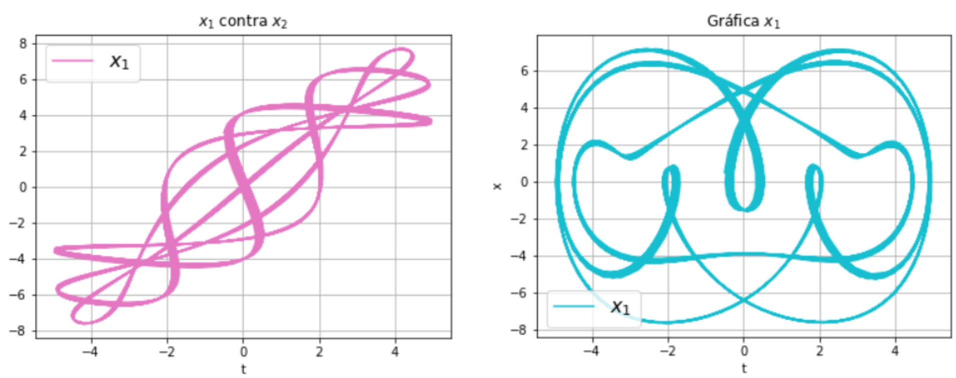
Las gráficas obtenidas en este ejemplo fueron:

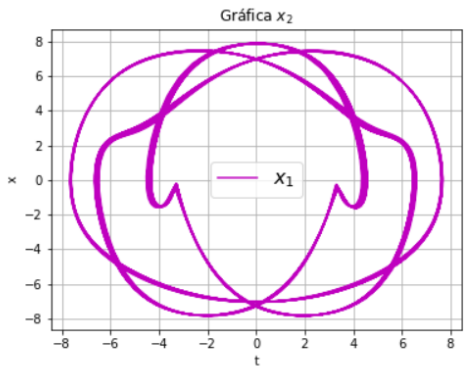




Ejemplo 3.2 *Asume m_1=m_2=1. Describe el movimiento para las constantes de resorte k_1=0.4 y k_2=1.808, coeficientes de amortiguamiento δ_1=0 y δ_2=0, coeficientes no lineales μ_1=-1/6 y μ_2=-1/10, con condiciones iniciales ($x_1(0)$, $\dot{x}_1(0)$, $x_2(0)$, $\dot{x}_2(0)$)=(-0.5,1/2,3.001,5.9)*

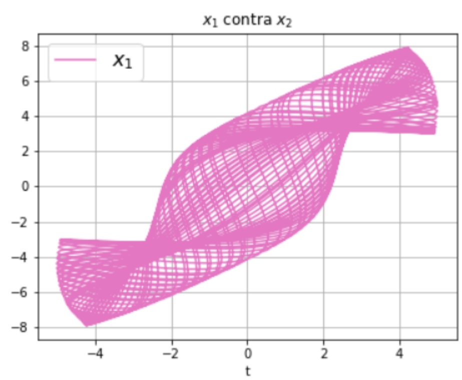
En este ejemplo no tenemos casi ningún cambio con el ejemplo anterior, solo modificamos las condiciones iniciales. Aún así, las gráficas que obtenemos como resultado son completamente distintas a las anteriores.

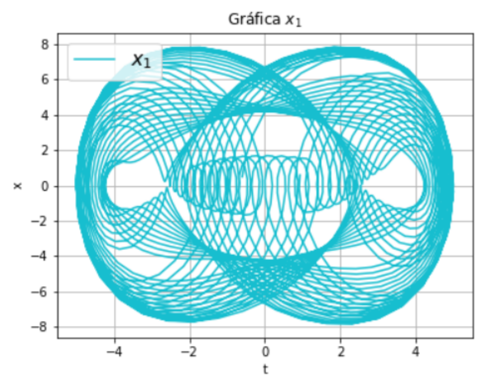


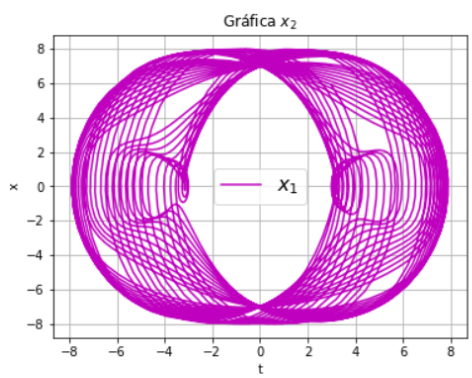


Ejemplo 3.3 *Asume m_1=m_2=1. Describe el movimiento para las constantes de resorte k_1=0.4 y k_2=1.808, coeficientes de amortiguamiento δ_1=0 y δ_2=0, coeficientes no lineales μ_1=-1/6 y μ_2=-1/10, con condiciones iniciales ($x_1(0)$, $\dot{x}_1(0)$, $x_2(0)$, $\dot{x}_2(0)$)=(-0.6,1/2,3.001,5.9)*

En este ejemplo, el cambio que se tiene es aún más delicado, cambiando solo amente una condición inicial del valor -0.5 a -0.6, dejando el resto del problema exactamente igual. Lo que inicialmente se piensa con este tipo de cambios sería que el cambio al sistema sería mínimo, pero al observar las gráficas observamos que no importa si a la condición inicial se le ha modificado bastante o, a primera instancia, de forma insignificante. El sistema no lineal se vuelve sumamente delicado.







2.4 Añadiendo Forzamiento

Añadir una fuerza externa al modelo es algo sencillo, inclusive agregando fuerzas diferentes a cada masa. Supongamos que asumimos un forzamiento senoidal simple, nuestro modelo es modificado a

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

$$m_2\ddot{x}_2 = -\delta_2\dot{x}_2 - k_2(x_2 - x_1) + \mu_2(x_2 - x_1)^3 + F_2\cos\omega_2 t$$

El rango de movimientos para modelos de fueras no lineales es muy amplio, las condiciones en las cuales estos movimientos ocurren siendo nada fáciles de plantear. Podemos encontrarnos con resonancia no linear, soluciones armónicas, soluciones sub-armónicas y ciclos límites en el plano fase. Al cambiar la ecuación de la cual nos basamos para realizar las soluciones, volvemos a modificar las primeras dos celdas.

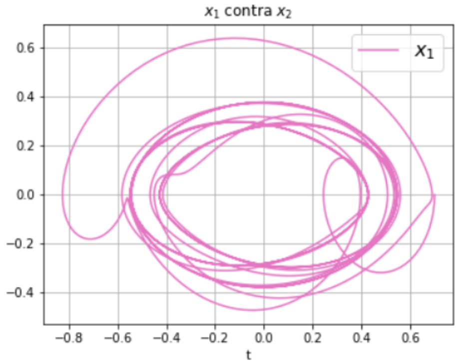
```
#Ejemplo 4.1
import numpy as np
def vectorfield(w, t, p):
    """
    Definimos las ecuaciones diferenciales para el sistema de doble masa-resorte.
    Arguments:
        w : Vector del estado de las variables
                  w = [x1,y1,x2,y2]
        t :  Tiempo
        p :  Vector de los parametros:
                  p = [m1,m2,k1,k2,L1,L2,b1,b2]
    """
    x1, y1, x2, y2 = w
    #Agregamos los coeficientes no lineales al vector de parámetros
    m1, m2, k1, k2, L1, L2, b1, b2, c1, c2, F1, F2, w1, w2 = p

    #Creamos f = (x1',y1',x2',y2')
    #Como vamos a usar una fuerza no lineal, debemos modificar la ecuación que trabajamos en la Actividad
 pasada
    f = [y1,
         (-b1 * y1 - k1 * (x1 - L1) + c1 * (x1 - L1) ** 3.0 - k2 * (x1 - x2 - L2) + c2 * (x1 - x2- L2) **
3.0 + F1 * np.cos(w1 * t)) / m1,
         y2,
         (-b2 * y2 - k2 * (x2 - x1 - L2) + c2 * (x2 - x1 - L2) ** 3.0 + F2 * np.cos(w2 * t)) / m2]
    return f
```

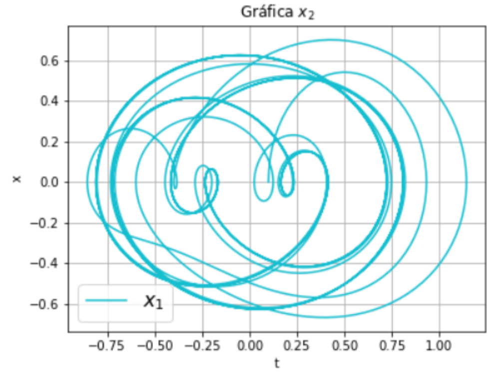
```python
#Usamos la funcion ODEINT para resolver las ecuaciones diferenciales definidas por el vector
from scipy.integrate import odeint
import numpy as np

#Valor de los parametros
# Masas:
m1 = 1.0
m2 = 1.0
# Constante del resorte
k1 = 2.0/5.0
k2 = 1.0
# Longitudes naturales
L1 = 0
L2 = 0
# Coeficientes de fricción
b1 = 1.0/10.0
b2 = 1.0/5.0

#Parametros de coeficientes no lineales
c1=1.0/6.0
c2=1.0/10.0

#Parametros de Fuerza
F1=1.0/3.0
F2=1.0/5.0

#Parametros de la frecuencia de la Fuerza
w1=1.0
w2=3.0/5.0

# Condiciones iniciales
# x1 and x2 son las pocisiones iniciales(contando la longitud de L), y y1 y y2 son las velocidades
x1 = 0.7
y1 = 0.0
x2 = 0.1
y2 = 0.0

# Parametros de la ED
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 170.0
numpoints = 2250

#Creamos los valores del tiempo, entre mas puntos damos, mejores se veran las graficas.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]

# Ponemos a las variables en un vector.
#Agregamos los coeficientes no lineales
p = [m1, m2, k1, k2, L1, L2, b1, b2, c1, c2, F1, F2, w1, w2 ]
w0 = [x1, y1, x2, y2]

# Llamamos a la funcion para resolver la ED
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

with open('ejemplo4.1.dat', 'w') as f:
    # Imprimimos en el documento la solución
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], w1[2], w1[3], file=f)
```

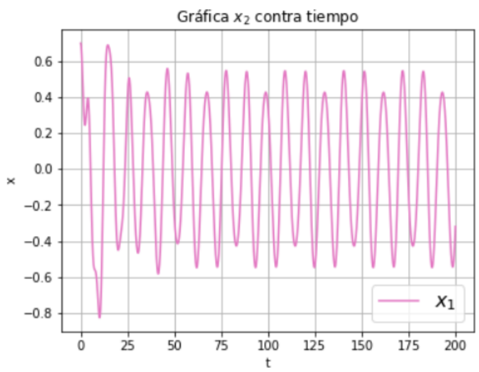
Ejemplo 4.1 *Asume m_1=m_2=1. Describe el movimiento para las constantes de resorte k_1=2/5 y k_2=1.0, coeficientes de amortiguamiento δ_1=1/10 y δ_2=1/5, coeficientes no lineales μ_1=1/6 y μ_2=1/10, amplitud de forzamiento F_1=1/3 y F_2=1/5 y frecuencias de forzamiento ω_1=1 y ω_2=3/5 con condiciones iniciales (x_1(0), $\dot{x}_1$(0), x_2(0), $\dot{x}_2$(0))=(0.7,0,0.1,0)*

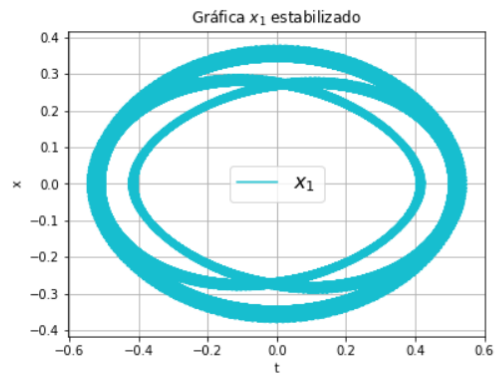
Al contar con un coeficiente de amortiguamiento, se espera observar un comportamiento diferente para los valores pequeños de t y comportamiento de estado estable para sus valores grandes. Por lo tanto, podemos esperar observar un ciclo límite en el plano fase para x_1 y x_2.

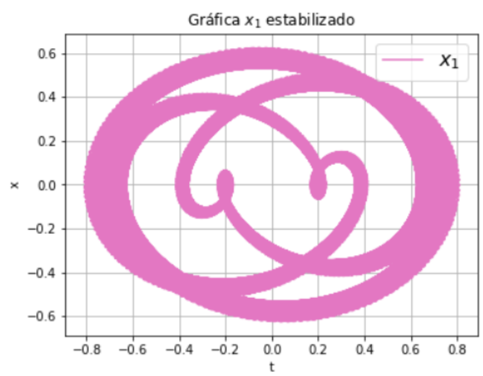
Las gráficas resultantes fueron:









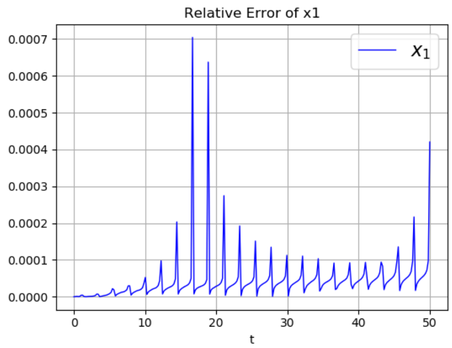


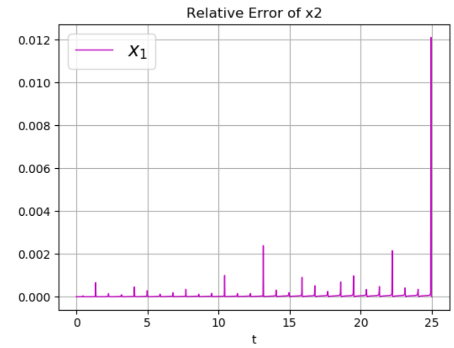
3 Errores Relativos

Al utilizar *Python* para resolver sistemas de ecuaciones lineales, estamos recurriendo a métodos numéricos para encontrar el resultado. Al recurrir a métodos numéricos estamos sometiendo nuestro resultado a cierto error, pues estamos aproximando y no obteniendo el valor real como se lograría de manera analítica. Dependiendo del número de particiones que llevemos a cabo podemos acercarnos al valor real reduciendo el valor del error a uno casi insignificante.

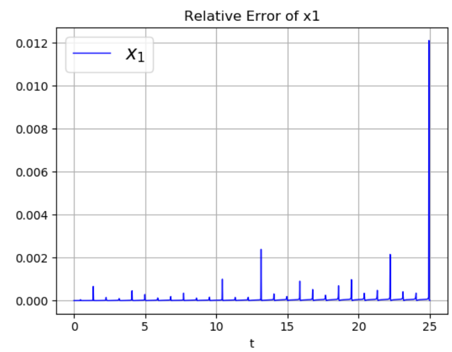
Para los ejemplos *Ejemplo 2.1* y *Ejemplo 2.2* podemos encontrar el error relativo del resultado analítico restando el resultado numérico creado en *Python* y graficarlo dependiente del tiempo. Como en el *Ejemplo 2.3* y *Ejemplo 2.4* no tenemos el resultado analítico, no podemos conseguir el error relativo.

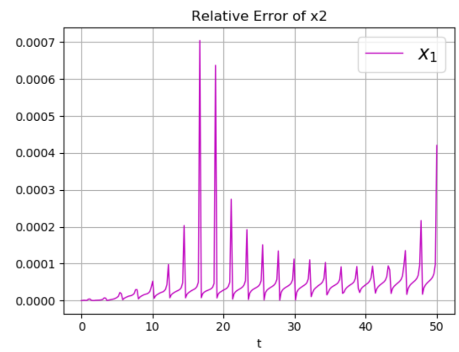
Las gráficas del error relativo en el *Ejemplo 2.1* al tener 250 particiones fueron:





Mientras que las gráficas del error relativo en el *Ejemplo 2.2* al tener 1250 particiones fueron:





En ambos casos se observan errores pequeños, obteniendo en las primeras dos gráficas un error del orden de $\times 10^{-4}$, mientras que se obtuvo en las últimas dos un error del orden de $\times 10^{-2}$.

4 Conclusión

Al utilizar herramientas como *JupyerLab*, el manejo de código que realiza operaciones tan potentes como lo son las soluciones de ecuaciones diferenciales de cuarto orden se vuelven más accesibles para el público, facilitando su manejo. Podemos encontrar soluciones de ecuaciones lineales de cualquier grado al tomar como variables sus condiciones iniciales como la fricción, amortiguamiento, masa y constantes de resorte. Incluso podemos resolver ecuaciones lineales con coeficientes no lineales y forzamiento, los cuales vuelven al sistema en uno más delicado por lo cual su manejo debe ser muy cuidadoso.

Los resultados que se obtienen son siempre en base a métodos numéricos, por lo cual se debe de tener en consideración que dependiendo del número de puntos o particiones que se le designe al código se estará más cerca al valor real y, por lo tanto, se tendrá un error pequeño.

5 Bibliografía

- Temple H. Fay, Sarah Duncan Graham (2003) Coupled Spring Equations. Int. J. Educ. Math. Sci. Tech.. Vol. 34, No. 1, pp. 65-79.

- SciPy Cookbook. Retrieved March 18, 2018.
  $http://scipy-cookbook.readthedocs.io/items/CoupledSpringMassSystem.html$

6 Apéndice

1. **¿Qué más te llama la atención de la actividad completa? ¿Que se te hizo menos interesante?** Lo fácil que es programar un sistema que me parece algo complicado plantear, en especial las partes que no son lineales. En general siento que todo era algo importante que debía prestarse atención, no hubo algo que me pareciera poco interesante.

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, pues como había mencionado en la práctica anterior, solo vimos brevemente la solución de resortes acoplados en nuestro curso de Mecánica II, por lo cual no se mencionaron fenómenos de este tipo.

3. **¿Qué propondrías para mejorar esta actividad? ¿Te ha parecido interesante este reto?** Un poco de explicación en el comportamiento que tenemos de las gráficas, pues en el documento no se explica bien qué ocurre con algunas, o porque se mueve de esa forma.

4. **¿Quisieras estudiar más este tipo de fenómenos no lineales?** Me gustaría, pero primero quisiera tener un poco más de apoyo en clase para entender su programación.