Actividad 6

Isaac Neri Gómez Sarmiento

28 de marzo, 2018

1 Introducción

En esta actividad, analizamos el movimiento de 2 resortes acoplados, es decir 2 resortes y 2 pesos conectados en serie colgando de un techo. Se utilizó el artículo *Coupled spring equations* de Temple H. Fay y Sarah Duncan para obtener las soluciones a las ecuaciones diferenciales que se construyen para modelar el sistema. Estas soluciones son funciones de desplazamiento que dependen del tiempo.

Se analizaron 4 casos en donde se conocen parámetros como las constantes de los resortes, masas, longitudes naturales, desplazamientos y velocidades iniciales. Las soluciones exactas se proporcionan en los 2 primeros casos, pero no en los 2 últimos casos. En todos los casos se determinaron soluciones numéricas usando la biblioteca de python **scipy.integrate**, específicamente la función **odeint**.

2 Modelo de resortes acoplados

La siguiente imágen representa gráficamente el sistema de resortes acoplados.

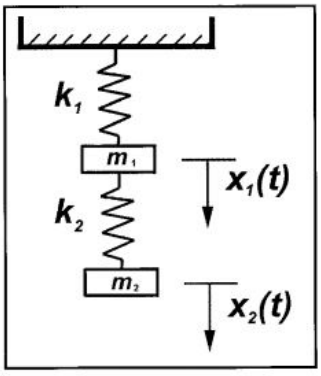


Figure 1: Sistema de resortes acoplados

Se tiene un resorte cuya constante es k_1 y éste está colgando de un techo. En la parte inferior del resorte se encuentra una masa m_1, de donde cuelga un resorte con constante k_2. En la parte inferior del segundo resorte cuelga una pesa de masa m_2. Se le permite al sistema llegar al reposo en equilibrio para poder medir apartir de ahí los posteriores desplazamientos de los centros de masa de cada pesa, los cuales se denotan como $x_1(t)$ y $x_2(t)$.

Las ecuaciones diferenciales se obtuvieron a partir de una sumatoria de fuerzas.

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

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

Figure 2: Sumatoria de fuerzas en cada resorte

De esta manera obtenemos un par de ecuaciones diferenciales de segundo orden. Para encontrar una ecuación para x_1 que no involucre x_2, despejamos a x_2, obteniendo:

$$x_2 = \frac{m_1 \ddot{x}_1}{k_2} + \frac{k_1 + k_2}{k_2} x_1$$

Sustituyendo x_2 en la segunda ecuación diferencial, y simplificando, obtenemos:

$$m_1 m_2 x_2^{(4)} + (m_2 k_1 + k_2(m_1 + m_2))\ddot{x}_2 + k_1 k_2 x_2 = 0$$

Por lo tanto el movimiento de la primera pesa está determinado por esta ecuación diferencial lineal de cuarto orden. Para encontrar una ecuación que solo involucre x_2, despejamos a x_1 de la segunda ecuación diferencial original:

$$x_1 = \frac{m_2}{k_2} \ddot{x}_2 + x_2$$

y sustituimos en la primera ecuación diferencial original, produciendo la ecuación:

$$m_1 m_2 x_2^{(4)} + (m_2 k_1 + k_2(m_1 + m_2))\ddot{x}_2 + k_1 k_2 x_2 = 0$$

Podemos ver que tiene la misma estructura que la ecuación que describe al primer resorte.

Si consideramos que las masas valen 1 kg de cada pesa, podemos simplificar la ecuación y determinar la ecuación característica que nos permite obtener las soluciones para $x_1(t)$ y $x_2(t)$.

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

cuyas raíces son:

$$\pm\sqrt{-\tfrac{1}{2}k_1 - k_2 \pm \tfrac{1}{2}\sqrt{k_1^2 + 4k_2^2}}$$

2.1 Ejemplo 2.1

Describir el movimiento para las constantes de los resortes $k1 = 6$ y $k2 = 4$ con condiciones iniciales:

$$(x_1(0),\ \dot{x}_1(0),\ x_2(0),\ \dot{x}_2(0)) = (1, 0, 2, 0)$$

En general se utilizó el mismo código para resolver los 4 ejemplos, solo que con algunas modificaciones en los parámetros, según el caso.

Primeramente, se define la función vectorfield que define las ecuaciones diferenciales para el sistema de resortes acoplados.

Las entradas son: w, t, p

- **w** es el vector que contiene las variables de posición (xi), y las de velocidad (vi) para cada uno de los resortes.
- **t** es el tiempo

- **p** es el vector que contiene los parámetros como la masa, la constante de los resortes, longitudes iniciales y constantes de amortiguamiento.

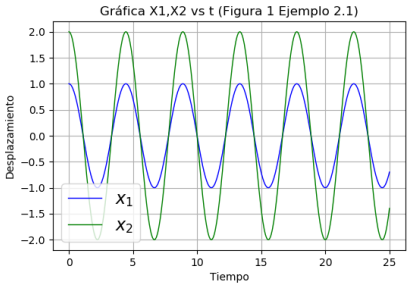
```
def vectorfield(w, t, p):
    x1, y1, x2, y2 = w
    m1, m2, k1, k2, L1, L2, b1, b2 = p

    f = [y1,
         (-b1 * y1 - k1 * (x1) + k2 * (x2 - x1)) / m1,
         y2,
         (-b2 * y2 - k2 * (x2 - x1)) / m2]
    return f
```

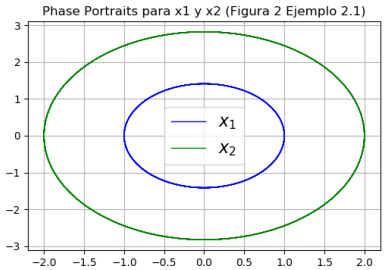
Se importan las bibliotecas numpy y scipy. De la biblioteca scipy se importa la función "odeint" para resolver las ecuaciones diferenciales.

```
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 = 25
numpoints = 500

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

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_springs1.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)
```

Al haber resuelto la ecuación, se procedió a graficar los desplazamientos x1 y x2 contra el tiempo y los phase portraits para x1 y x2.

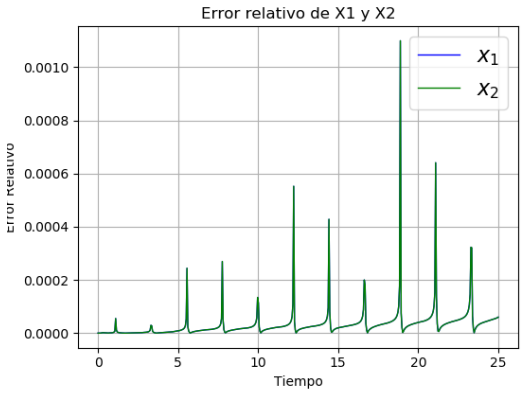




La solución analítica que se obtuvo en el artículo está dada por

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

La gráfica del error relativo entre la solución aproximada y la solución exacta es la siguiente:



2.2 Ejemplo 2.2

Describa el movimiento para las constantes de resorte $k_1 = 6$ y $k_2 = 4$ con las condiciones iniciales:

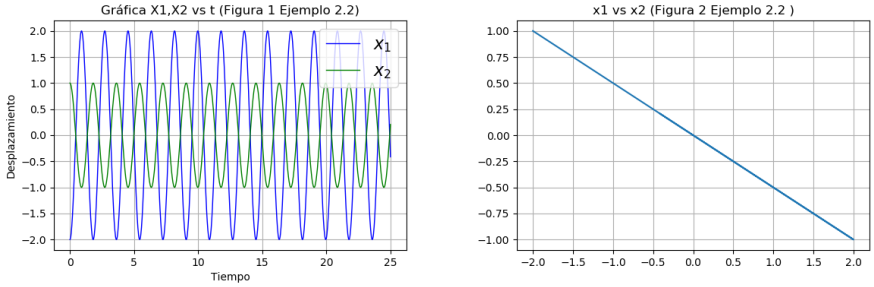
$$(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (-2, 0, 1, 0)$$

A continuación se muestra el segmento de código que se cambió del código anterior:

```
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
numpoints = 500

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

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],
 np.abs((w1[0]-(-2*np.cos(2*np.sqrt(3)*t1)))/(-2*np.cos(2*np.sqrt(3)*t1))),
 np.abs((w1[2]-(np.cos(2*np.sqrt(3)*t1)))/(np.cos(2*np.sqrt(3)*t1))), file=f)
```

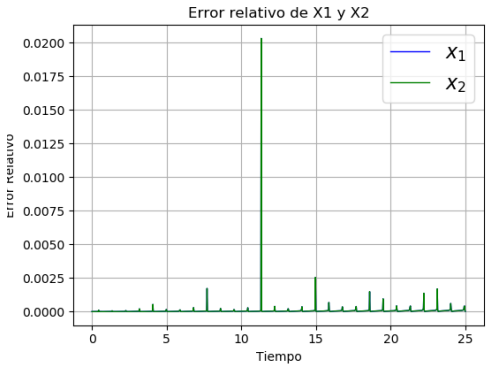
Las gráficas resultantes son las siguientes:



La solución analítica está determinada por:

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

La gráfica del error relativo de la solución analítica y la solución aproximada es la siguiente:



2.3 Ejemplo 2.3

Describa el movimiento para las constantes de resprte $k_1 = 0.4$ y $k_2 = 1.808$ con las siguientes condiciones iniciales:

$$(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (1/2, 0, -1/2, 7/10)$$

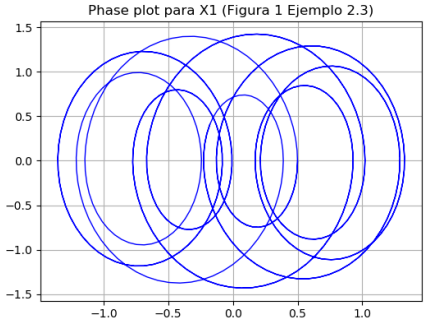
En este ejemplo y en el siguiente no se proporcionan las soluciones analíticas de las ecuaciones diferenciales, por lo que no se obtuvo una gráfica para el error relativo.

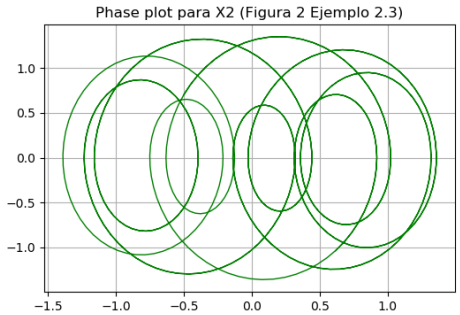
El segmento de código que fue modificado quedó de la siguiente manera:

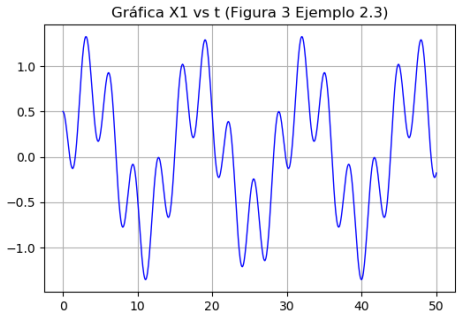
```
from scipy.integrate import odeint
import numpy as np

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

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = 1./2.
y1 = 0.0
x2 = -1./2.
y2 = 7./10.

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50
numpoints = 800

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

t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]
```

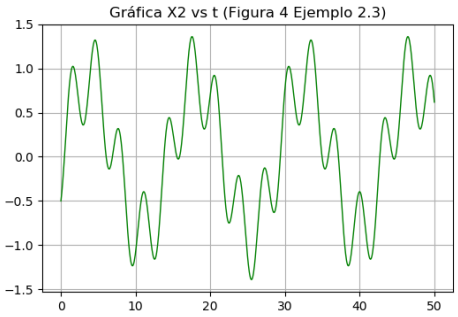
```
# Pack up the parameters and initial conditions:
p = [m1, m2, k1, k2, L1, L2, b1, b2]
w0 = [x1, y1, x2, y2]

# 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)
```

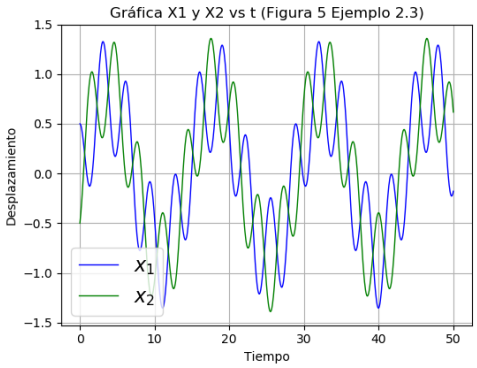
Las gráficas que se obtuvieron son las siguientes:

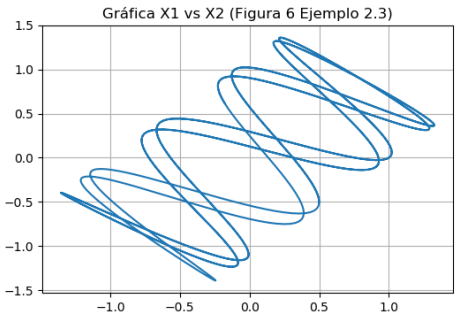












2.4 Ejemplo 2.4

Asuma que $m_1 = m_2 = 1$. Describa el movimineto para las constantes de resorte $k_1 = 0.7$ y $k_2 = 1.808$, coeficientes de amortiguamiento $b_1 = 0.1$ y $b_2 = 0.2$, con las condiciones iniciales:

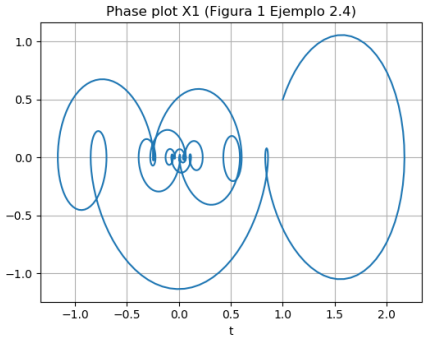
$$(x_1(0), \dot{x}_1(0), x_2(0), \dot{x}_2(0)) = (1, 1/2, 2, 1/2)$$

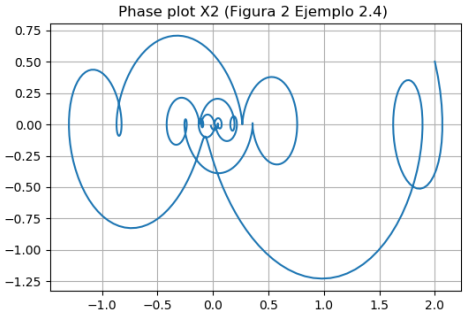
La diferencia entre este ejemplo y los anteriores, es que ahora se toman en cuenta los coeficientes de amortiguamiento.

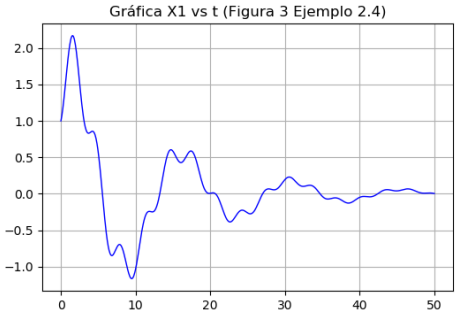
El segmento de código queda como sigue:

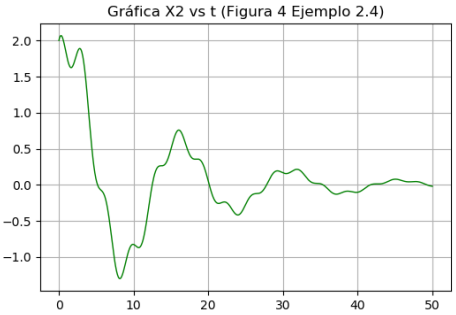
```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint

# 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
y1 = 1./2.
x2 = 2.
y2 = 1./2.

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50
numpoints = 800

# 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)
```

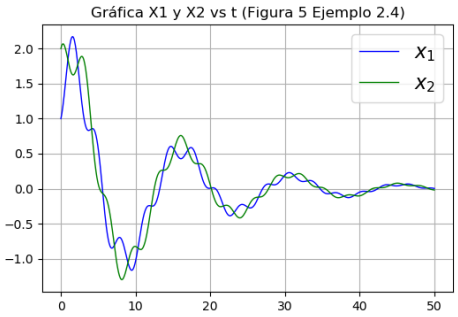
Se presentan las gráficas obtenidas:

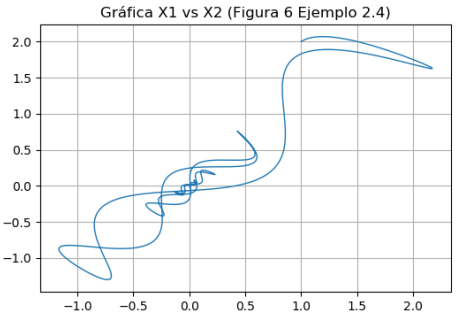












3 Conclusión

Esta actividad fue un primer acercamiento al modelado de un sistema físico, sistema de resortes acoplados, usando la ayuda de algoritmos computacionales para resolver ecuaciones diferenciales ordinarias.

Algunas observaciones se pudieron rescatar, como el hecho de que entre más parámetros se tomen en cuenta (como las constantes de amortiguamiento en el último ejemplo), más caótico se vuelve el movimiento y se ve reflejado en la simetría de las gráficas. Los errores relativos representados en gráficas en los dos primeros ejemplos, muestran que no hay mucha diferencia entre la solución analítica y la aproximada, siendo la máxima diferencia no más del 2%.

4 Apéndice

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

   Si me pareció interesante, en especial que la solución aproximada haya sido muy fiel a la analítica. Me llama la atención la forma de las gráficas. No hubo nada que no me gustara, lo único que no entendí fue la interpretación de la gráficas *phaseplot*.

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

   Me pareció suficiente, ya que no tuvimos que resumir mucho texto y además, que estamos familiarizados con el tema.

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

   Es muy similar a Jupyter Notebook, con la diferencia que se pueden acomodar las celdas en el orden que uno quiere después de haberlas creado o se puede acceder fácilmente a las imágenes o archivos guardados.

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

   Si había visto otros métodos de integración numérica. En cálculo 2 y Fortran ví el método de los trapecios. En análisis numérico, además del de trapecios, vi el de Simpson 1/3 y 3/8.

   Para programar, el de trapecios es más fácil, ya que en el de Simpson debes considerar casos en el que debes usar 1/3 y 3/8 dependiendo del número de particiones.

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

   No que yo recuerde. Solo para un sistema de masas con un solo resorte.

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

   Habría sido mejor que hubiera consistido en un solo problema, en el cual por equipos hubieramos obtenido experimentalmente las condiciones iniciales, las masas, las constantes de los resortes y los coeficientes de amortiguamiento. De esta manera, proseguiriamos a obtener nosotros las soluciones analíticas a las ecuaciones diferenciales y por último la solución con integración numérica.

5 Bibliografía

Temple H. Fay and Sarah D. Graham. Coupled Spring Equations. Recuperado el 12 de Marzo 2018 de: http://math.oregonstate.edu/~gibsonn/Teaching/MTH323-010S15/Supplements/coupled_spring.pdf

Sci-py Cookbook. Coupled spring-mass system. Recuperado el 12 de Marzo 2018 de: http://scipy-cookbook.readthedocs.io/items/CoupledSpringMassSystem.html