Actividad 6:
Sistema de resortes acoplados

Melissa Matrecitos Avila

20 de Marzo de 2018

1 Introducción

El siguiente reporte corresponde a la actividad 6 del curso de Física Computacional 1, en la cual se enfocó en la modelación de fenómenos físicos, apoyados con Python, en especial con un sistema de resortes acoplados. Esto se logro mediante la aplicación de métodos numéricos, realizando las gráficas los resultados y comparando. cuando era posible, con la solución exacta.

Para inciar con el reporte se muestra una pequeña síntesis sobre las secciones 1 y 2 del artículo "Coupled spring equations" de Temple H. Fay y Sarah Duncan Graham junto con las secciones de código utilizadas para resolver el problema numericamente. También se incluye la comparación de los resultados numéricos con los exactos mediante la gráfia del error relativo. Por último se presentan las secciones de conclusión, bibliografía y apéndice.

2 Síntesis

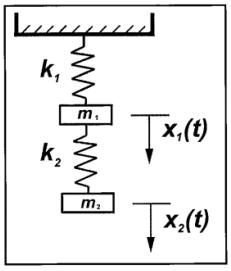
2.1 Introducción

El estudio clásico de las ecuaciones diferenciales está cambiando rápidamente, esto se debe en particular a la amplia disponibilidad de algoritmos numéricos de alta potencia y capacidades gráficas con un énfasis en las ecuaciones no lineales

En este artículo, se investiga el viejo problema de dos resortes y dos pesos unidos en serie, colgando del techo. 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 sustituir una ecuación por otra, el movimiento de cada peso se puede demostrar que está determinado por una ecuación diferencial lineal de cuarto orden.

Con las ecuaciones mencionadas se pueden estudiar algunos fenómenos más sobresalientes de este tipo de movimientos, como lo son la periodicidad, la amplitud, la fase, la sensibilidad a las condiciones iniciales y muchos otros conceptos, tan solo modificando los parámetros en el modelo.

2.2 El modelo de resorte acoplado



El modelo consiste en dos resortes y dos pesos. Un resorte con constante 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 con una constante de resorte k_2. En la parte inferior de este segundo resorte, se adjunta un peso de masa m_2.

Considerando el sistema desde el equilibrio, medimos el desplazamiento del centro de masa de cada peso como una función del tiempo, y denotamos estas medidas por x1 (t) y x2 (t), respectivamente.

2.2.1 Asumiendo la Ley de Hooke

Bajo la suposición de pequeñas oscilaciones, las fuerzas de restauración son de la forma $-k_1 l_1$ y $-k_2 l_2$ donde l_1 y l_2 son los alargamientos (o compresiones) de los dos resortes.

Dado que la masa superior está unida a ambos resortes, hay dos fuerzas de restauración que actúan sobre ella: una fuerza de restauración hacia arriba $k1l1$ ejercida por el alargamiento (o compresión) $x1$ del primer resorte; una fuerza hacia arriba $k_2(x_2 - x1)$ desde la resistencia del segundo muelle a ser alargada (o comprimida) en la cantidad $x_2 - x1$. La segunda masa solo "siente" la fuerza de restauración desde el alargamiento (o compresión) del segundo resorte. Si asumimos que no hay fuerzas de amortiguamiento presentes, entonces la Ley de Newton implica que las dos ecuaciones que representan los movimientos de los dos pesos son:

$$m_1 \ddot{x_1} = -k_1 x_1 - k_2(x_1 - x_2)$$

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

Para encontrar una ecuación para x_1 y x_2 que no dependan una de la otra, se resuelve la ecuación de x_2 y se sustituye en la de x_1, obteniendo las ecuaciones:

$$m_1 m_2 x_1^{(4)} + (m_2 k_1 + k_2(m_1 + m_2))\ddot{x_1} + k_1 k_2 x_1 = 0$$
$$m_1 m_2 x_2^{(4)} + (m_2 k_1 + k_2(m_1 + m_2))\ddot{x_2} + k_1 k_2 x_2 = 0$$

2.2.2 Algunos ejemplos con masas idénticas

Considerando un sistema normalizado con masas iguales, esto es $m_1 = m_2 = 1$, y asumiendo que no hay fricción ni fuerzas externas, la 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 con constantes* $k_1 = 6$ *y* $k_2 = 4$ *con condiciones iniciales* $(x_1(0), \dot{x_1}(0), x_2(0), \dot{x_2}(0))$*=(1,0,2,0).*

Resolviendo analíticamente encontramos que las soluciones para x_1 y x_2 son:

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

El movimiento es sincronizado y las ondas se mueven fase una con la otra, teniendo el mismo periodo de movimiento pero con distintas amplitudes. Como el movimiento es periódico simple, las fases de x_1 y x_2 forman elipses.

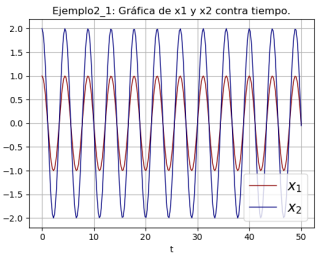
A continuación se muestra el procedimiento correspondiente al método numérico utilizado en todos los ejemplos (variando solo las constates indicadas), este se logro utilizando Python, obteniendo los datos y guardándolos en un archivo para después ser utilizados en las gráficas como se ha hecho en prácticas pasadas:

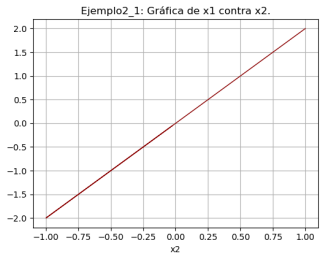
```
def vectorfield(w, t, p):
    """
    Defines the differential equations for the coupled spring-mass system.

    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
```

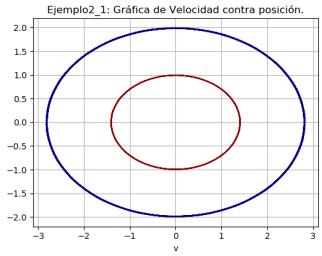
```
# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint
import numpy as np
import math

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.0
# Spring constants
k1 = 6.0
k2 = 4.0
# Natural lengths
L1 = 0.0
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('Ejemplo2_1.dat', 'w') as E21:
    # 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=E21)
```

```
# 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, e1, e2 = loadtxt('Ejemplo2_1.dat', unpack=True)

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

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

plot(t, x1, 'darkred', linewidth=lw)
plot(t, x2, 'navy', linewidth=lw)

legend((r'$x_1$', r'$x_2$'), prop=FontProperties(size=16))
title('Ejemplo2_1: Gráfica de x1 y x2 contra tiempo.')
savefig('Ejemplo2_1Oscilaciones.png', dpi=100)
```

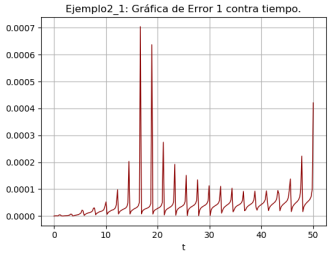
Las graficas obtenidas fueron:

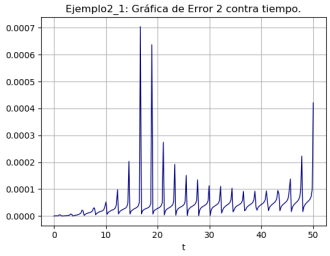






Cómo el articulo presenta las soluciones analíticas, es posible calcular el error relativo para la posición de cada masa, obteniendo así las gráficas del error realtivo contra el tiempo:





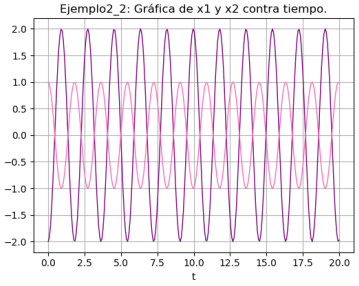
Ejemplo 2.2:*Describe el movimiento para resortes con constantes* $k_1 = 6$ *y* $k_2 = 4$ *con condiciones iniciales* $(x_1(0), \dot{x_1}(0), x_2(0), \dot{x_2}(0))$*=(-2,0,1,0).*

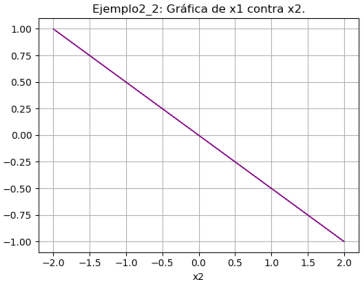
Resolviendo analíticamente encontramos que las soluciones para x_1 y x_2 son:

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

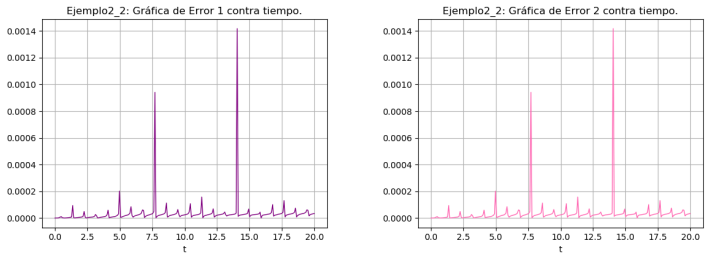
Al igual que el ejemplo anterior tienen el mismo periodo de movimiento y distintas amplitudes, solo que ahora mientras una se mueve en una dirección, la otra lo hace en sentido contrario. Así que realizando los cambios necesarios (como se muestra en la imagen) se obtuvieron las siguientes gráficas:

```
# 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 = 0.0
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 = 20.0
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('Ejemplo2_2.dat', 'w') as E22:
    # 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(3.0)*t1))))/(-2*(np.cos(2
*np.sqrt(3.0)*t1)))), np.abs((w1[2]-(np.cos(2*np.sqrt(3.0)*t1)))/(np.cos(2*np.sqrt(3.0)*t1))),file=E22)
```





Cómo en el ejemplo anterior, el articulo también presenta las soluciones analíticas, por lo que también es posible calcular el error relativo para la posición de cada masa, obteniendo así las gráficas del error relativo contra el tiempo:

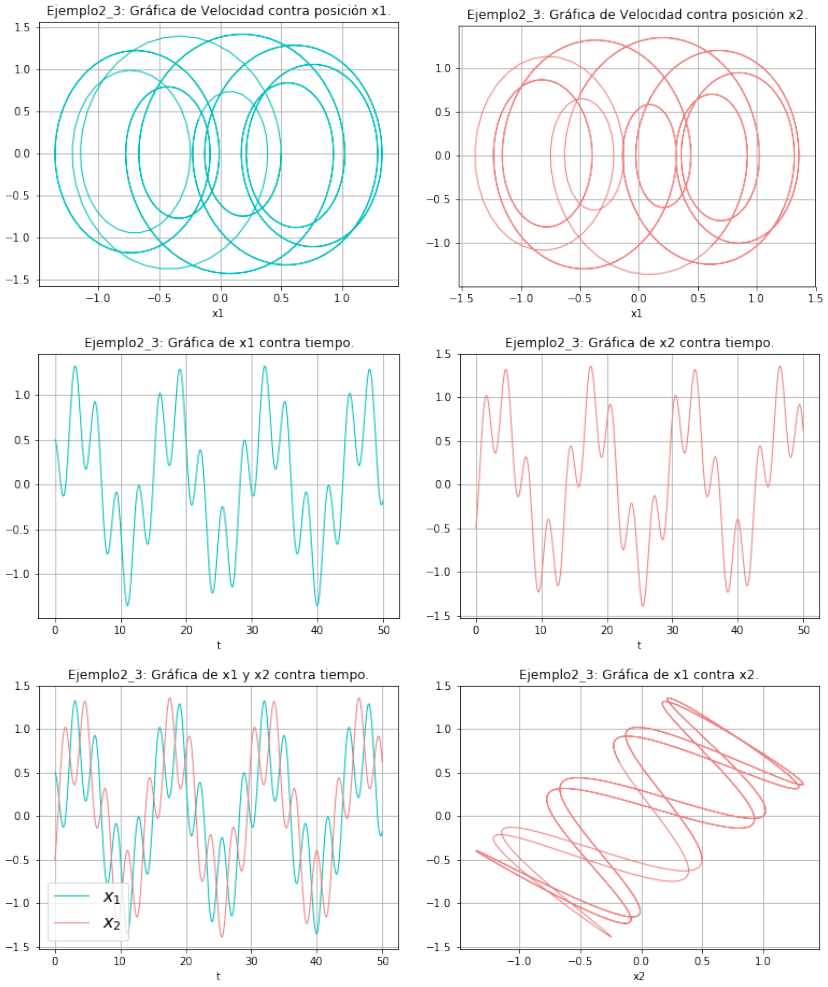


Ejemplo 2.3:*Describe el movimiento para resortes con constantes* $k_1 = 0.4$ *y* $k_2 = 1.808$ *con condiciones iniciales* $(x_1(0), \dot{x_1}(0), x_2(0), \dot{x_2}(0))$*=(1/2,0,-1/2,7/10).*

En este ejemplo se nota como los valores de k_1 y k_2 determinan por completo el periodo y las condiciones iniciales solo afectan la amplitud y la fase de las soluciones. Este no cuenta con solución analítica, sin embargo aun se puede resolver con métodos numéricos:

```
# 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 = 0.5
y1 = 0.0
x2 = -0.5
y2 = 0.7

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 50.0
numpoints = 500

# 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 E23:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], w1[2], w1[3], file=E23)
```

Obteniendo las gráficas:



2.2.3 Amortiguamiento

EL tipo más común de amortiguamiento es el provocado por la viscosidad, donde la fuerza amortiguadora es proporcional a la velocidad. El amortiguamiento de la primera masa depende solo de su velocidad y no de la velocidad de la segunda masa y viceversa. Para agregar este fenómeno a las ecuaciones, se añaden los términos $-\delta_1\dot{x}_1$ y $-\delta_2\dot{x}_2$ a las ecuaciones correspondientes. Asumiendo que δ_1 y δ_2 son muy pequeños, el sistema queda:

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

Realizando el mismo proceso para encontrar las nuevas ecuaciones para x_1 y x_2 que no dependan una de la otra, obtenemos:

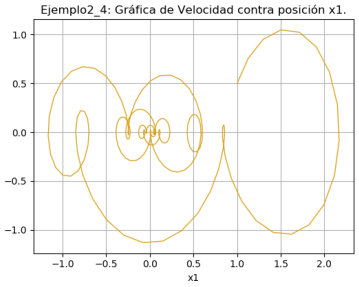
$$m_1 m_2 x_2^{(4)} + (m_1\delta_1 + m_2\delta_2)\dddot{x}_1 + (m_2 k_1 + k_2(m_1 + m_2))\ddot{x}_2 + k_1 k_2 x_2 = 0$$

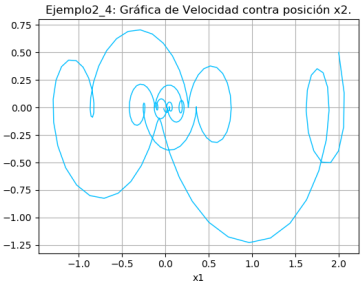
Ejemplo 2.4: *Asuma que* $m_1 = m_2 = 1$.*Describe el movimiento para resortes con constantes* $k_1 = 0.4$ *y* $k_2 = 1.808$, *coeficientes de amortiguamiento* $\delta_1 = 0.1$ *y* $\delta_2 = 0.2$ *con condiciones iniciales* $(x_1(0), \dot{x_1}(0), x_2(0), \dot{x_2}(0))$=*(1,1/2,2,1/2).*

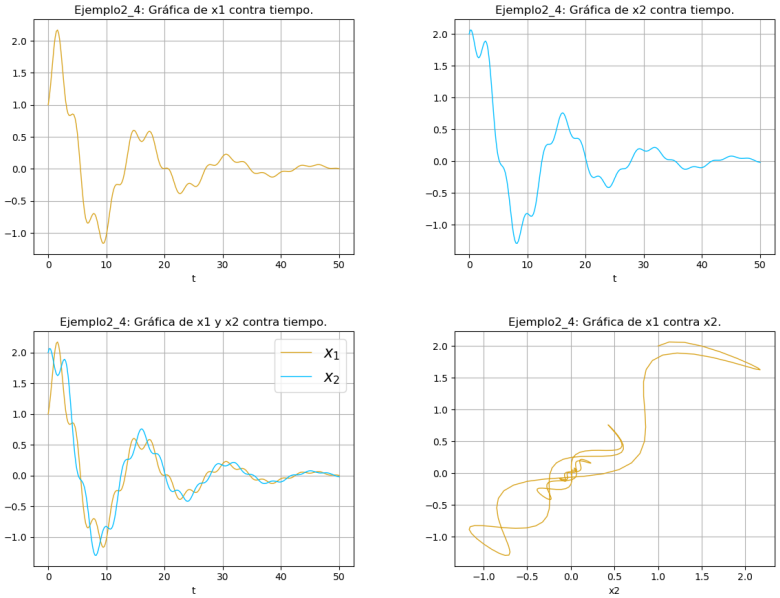
ebido al factor de amortiguamiento, la amplitud del movimiento disminuye conforme el tiempo avanza, lo cual se hace notorio en las gráficas de x1 contra tiempo y x2 contra tiempo.

```
# 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 = 0.5
x2 = 2.0
y2 = 0.5

# 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('Ejemplo2_4.dat', 'w') as E24:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print (t1, w1[0], w1[1], w1[2], w1[3], file=E24)
```

Obteniendo las gráficas:







3 Conclusión

Trabajar con el interfaz de Jupyter Lab durante esta actividad fue de gran ayuda, ya que permite una mejor organización con los archivos, sin mencionar que las imágenes se generan inmediatamente.

Respecto al tema con el que se trabajó, se había visto en Mecánica 2, pero me pareció más sencillo visto desde la perspectiva de la programación, aun que era la primera vez que trabajaba con ecuaciones diferenciales, su solución fue muy eficaz.

4 Bibliografía

- H. FAY, T., & DUNCAN GRAHAM, S. (2003). Coupled spring equations (pp. 65-79). Int. J. Math. Educ. Sci. Technol. Recuperado de `http://math.oregonstate.edu/~gibsonn/Teaching/MTH323-010S15/Supplements/coupled_spring.pdf`

5 Apéndice

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

   Me pareció muy interesante ya que era algo completamente nuevo para mi, aun que el código ya estaba escrito, entender como funcionaba fue muy

entretenido.

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

   Conforme fui realizando la actividad me pareció cada vez más adecuado.

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

   Me pareció mucho mejor que Jupyter Notebook, debido a que era más fácil manejar los archivos y las funcioes que se pueden utilizar son más atractivas.

4. Respecto al uso de funciones de SciPy, ¿ya habías visto integración numérica en tus cursos anteriores? ¿Cuál es tu experiencia?. Sí lo habaía visto pero era muy distinto, se habían abordado los temas de integración numérica mediante el método de trapecios y Simpson, pero no como en esta actividad,

5. El tema de sistema de masas acopladas con resortes, ¿ya lo habías resuelto en tu curso de Mecánica 2? Se resolvió pero de una manera muy simple, en el curso en el que sí se aboró un poco más fue en el de Ecuaciones Diferenciales.

6. ¿Qué le quitarías o agregarías a esta actividad para hacerla más interesante y divertida? Así como está me parece muy bien.