# UNIVERSIDAD DE SONORA


DIVISIÓN DE CIENCIAS EXACTAS Y NATURALES

DEPARTAMENTO DE FÍSICA

**Física Computacional 1**

# Actividad 2: Utilizando Python para resolver un sistema de ecuaciones diferenciales ordinarias

**Profesor: Carlos Lizarraga Celaya**

**Alumno: Enrique Vizcarra Carrazco**

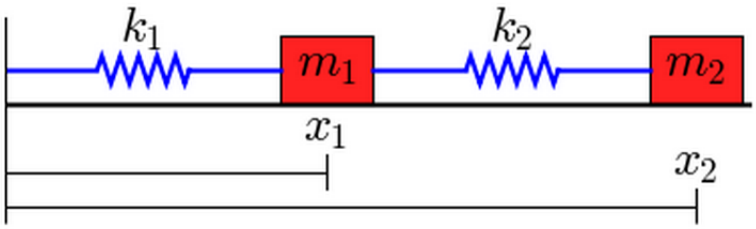
**fecha: 6 de octubre de 2014**

**Resumen**

Es el presente documento/ actividad se hablará de como utilizar Python para resolver un sistema de ecuaciones diferenciales ordinarias (EDO), en particular se ejemplifica el uso de la función Odeint dentro de la biblioteca scipy.integrate para resolver las ecuaciones de movimiento de un sistema masa-resorte acoplado.

# Introducción

Para mostrar el uso de como utilizar Python para resolver ecuaciones diferenciales, primero consideremos un sistema masa-resorte acoplado como muestra la siguiente imagen:




Dos objetos de masas $m1$ y $m2$ estan acoplados por medio de resortes, cuyas constantes son $k1$ y $k2$ respectivamente. El extremo izquierdo del resorte del lado izquierdo se encuentra fijo. Asumimos que las longitudes naturales de los resortes son $L1$ y $L2$ respectivamente.
Las masas se deslizan sobre una superficie tal que crea fricción, por lo cual hay dos coeficientes de fricción asociados $b1$ y $b2$.

# 1. Ecuaciones de Movimiento del Sistema

Las ecuaciones diferenciales para este sistemas son:

$$m1x1'' + b1x1' + k1(x1 - L1) - k2(x2 - x1 - L2) = 0$$
$$m2x2'' + b2x2' + k2(x2 - x1 - L2) = 0$$

Este es un par de ecuaciones diferenciales acopladas de segundo orden. Para resolver este sistema usaremos uno de los solucionadores de ecuaciones de EDO proporcionado por SciPy, primero se debe reducir estas ecuaciones a un sistema de EDO de primer orden. Para esto introduciremos las variables:

$$y1 = x1'; \quad y2 = x2'$$

Estas representan las velocidades de las masas.
Con un poco de álgebra, se puede reescribir el sistema original como el siguiente sistema de cuatro EDO de primer orden:

$$x1' = y1$$
$$y1' = (-b1y1 - k1(x1 - L1) + k2(x2 - x1 - L2))/m1$$
$$x2' = y2$$
$$y2' = (-b2y2 - k2(x2 - x1 - L2))/m2$$

Estas ecuaciones son ahora de forma tal que podemos implementarlas en Python.

# 2. Código Utilizado

Para resolver el sistema de ecuaciones se utilizaron los siguientes fragmentos de cóigo tomado del Cookbook de SciPy. el código esta divido en tres módulos.

## 2.1. Primero Módulo: Definiendo el Campo vectorial

La primera parte de el código define el lado derecho del sistema de ecuaciones también conocido como el campo vectorial. Los argumentos de la función que define el campo vectorial se encuentran configurados para se r usados con la función Odeint: El tiempo $t$ es el segundo argumento:

```
#
# two_springs.py
#
"""
This module defines the vector field for a spring-mass system
consisting of two masses and two springs.
"""


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
```

## 2.2. Segundo Módulo: Solucionador del sistema de ecuaciones

El segundo módulo utiliza Odeint para resolver el sisitema de ecuaciones para el conjuro de valores de los parámetros dados, condiciones iniciales, y tiempo del intervalo. El código

original imprime los puntos de la solución en la terminal, por lo que tuvo que ser adaptado con el fin de almacenarlos en un archivo nuevo con el fin de usarlos mas tarde.

```
#
# two_springs_solver.py
#
"""Use ODEINT to solve the differential equations defined by the vector field
in two_springs.py.
"""

from scipy.integrate import odeint
import two_springs

# Parameter values
# Masses:
m1 = 1.0
m2 = 1.5
# Spring constants
k1 = 8.0
k2 = 40.0
# Natural lengths
L1 = 0.5
L2 = 1.0
# Friction coefficients
b1 = 0.8
b2 = 0.5

# Initial conditions
# x1 and x2 are the initial displacements; y1 and y2 are the initial velocities
x1 = 0.5
y1 = 0.0
x2 = 2.25
y2 = 0.0

# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 10.0
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(two_springs.vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)

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

En el segmento anterior se modificaron las ultimas lineas, utilizando una función "file"para designar la creación de un nuevo archivo llamado "datoslistaa.dat" con el atributo W (para escritura), se definió a demás una función conteniendo las variables de la solución convertidas al tipo "string" (el atributo "rjust(20)" permite darle un formato a la variable tipo string, en este caso hace un justificado hacia la derecha con 20 espacios a consideración) además se declara la variable "da" para contener las variables de la solución y se usó el comando "writelines" para escribirlos en el nuevo archivo, al final se ordena cerrar el documento:

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

file = open("datoslistaa.dat", "w")
# Print the solution.
   # print t1, w1[0], w1[1], w1[2], w1[3]
for t1, w1 in zip(t, wsol):
    da = [str(t1).rjust(20), str(w1[0]).rjust(20), str(w1[1]).rjust(20),
     str(w1[2]).rjust(20),str(w1[3]).rjust(20),'\n']

    file.writelines(da)
file.close()
```

## 2.3. Tercer Módulo: Graficador

El último módulo permite graficar los datos generados por el segmento de código anterior usando Matplotlib, este también fué adaptado para que tomara lectura del los datos desde los archivos creados en el segmento anterior:

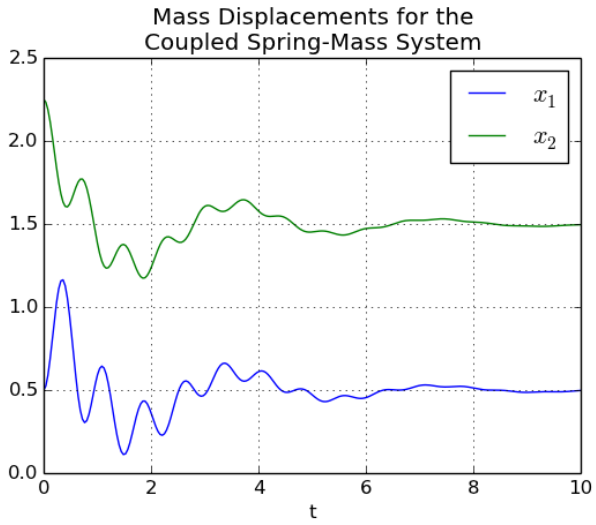
```
#
# two_springs_plot.py
#
"""Plot the solution that was generated by two_springs_solver.py."""
```

```
from numpy import loadtxt
from pylab import figure, plot, xlabel, grid, hold, legend, title, savefig
from matplotlib.font_manager import FontProperties


t, x1, xy, x2, y2 = loadtxt('datoslistaa.dat', unpack=True)

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

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('Mass Displacements for the\nCoupled Spring-Mass System')
savefig('two_springsBB.png', dpi=100)
```

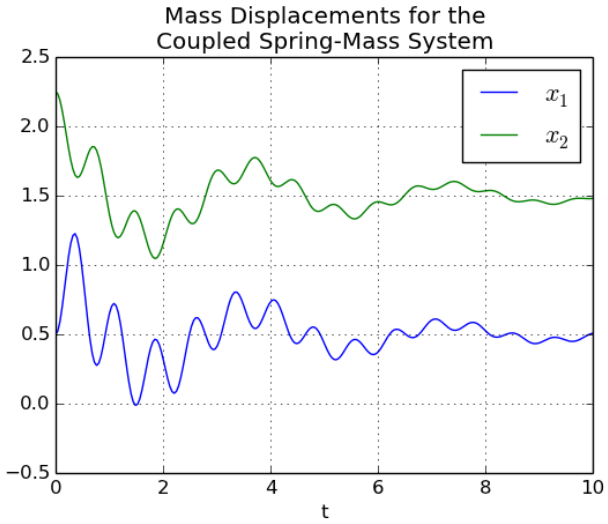
# Resultados

El resultado obtenido al modificar los parámetros para los coeficientes de fricción fueron tres gráficas de posición contra tiempo de amabas masas, las cuales se muestran acontinuación:

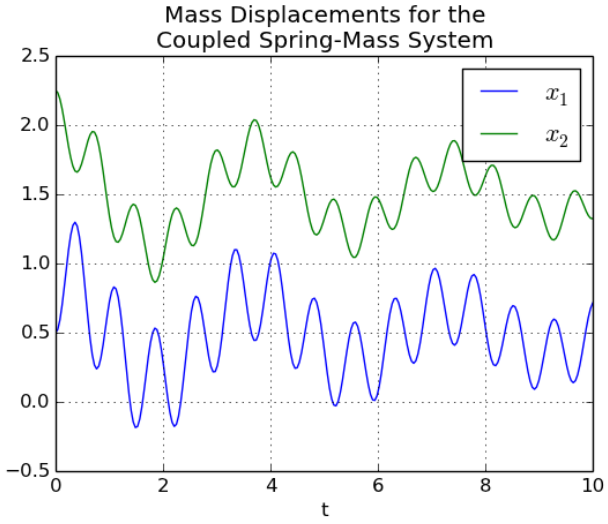
a) b1=1.6, b2=0.5




b) b1=0.8, b2=0.5



c) b1=0.0, b2=0.5


Mass Displacements for the
Coupled Spring-Mass System
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0
2
4
6
8
10
t
$x_1$
$x_2$