Actividad 9

Solución numérica de ecuaciones diferenciales ordinarias.

Jorge Benz Olguín Aguilar

División de Ciencias Exactas, Departamento de Física

Universidad de Sonora

29 de mayo de 2019

Al igual que otros lenguajes de programación de alto nivel como lo es C, C++ y Fortran, en el caso de Python cuenta con las bibliotecas NumPy y SciPy, las cuales se pueden utilizar para encontrar soluciones numéricas de ecuaciones diferenciales, tema importante en Análisis Numérico.

En este proyecto, resolveremos numéricamente un sistema de ecuaciones diferenciales ordinarias que describen el sistema que aparece en la figura de abajo. Intentaremos resolverlo numéricamente utilizando la función odeint de SciPy. Nos basaremos en las notas de Richard Fitzpatrick, de la Universidad de Texas que resuelve analíticamente el sistema arriba mencionado.[1]

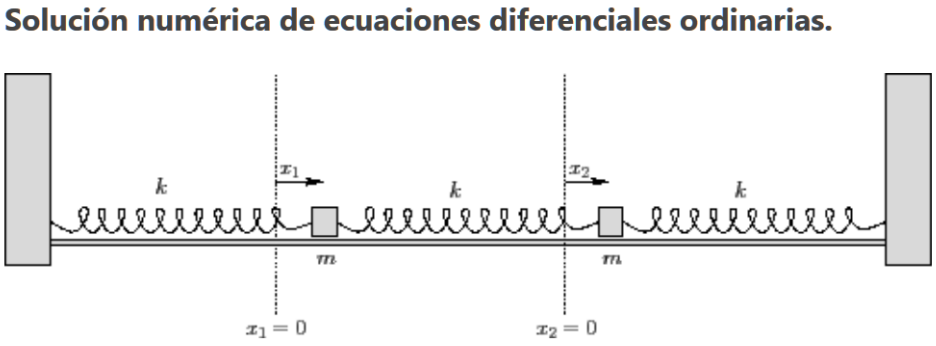

Figura 1:

Sistema masa resorte de dos masas acopladas. Consideremos un sistema mecánico de dos cuerpos de masa m acopladas por unos resortes k y no consideramos que exista fricción. El estado del sistema es convenientemente especificado para que tenga desplazamientos a la izquierda y a la derecha, x_1 y x_2 respectivamente, asumimos que $x_1 = x_2 = 0$ corresponde a la configuración de equilibrio del sistema en el cual los resortes no están extendidos. Las extensiones a la izquierda, centro y derecha de los resortes son x_1, $x_2 - x_1$ y $-x_2$, respectivamente. Las ecuaciones de movimiento de estas dos masas son

$$mx_1^{''} = -kx_1 + k(x_2 - x_1)$$

$$mx_2^{''} = -k(x_2 - x_1) + k(-x_2)$$

Estas son un par de ecuaciones diferenciales de segundo orden. Para resolver este sistema con uno de los ODE solvers de Scipy, nosotros debemos primero convertir este a un sistema de ecuaciones diferenciales de primer orden. Par eso introducimos dos variables

$$y_1 = x_1^{'}$$

Estas son las velocidades de las masas y a partir de aquí podemos reescribir las ecuaciones de segundo orden en un sistema de cuatro ecuaciones de primero:

$$x_1^{'} = y1$$

$$y_1^{'} = \frac{k(-2x_1 + L_1 + x_2 - L_2)}{m}$$

$$x_2^{'} = y_2$$

$$y_2^{'} = \frac{-k(x_2 - x_1 - L_2}{m}$$

Estas ecuaciones están en forma que python las puede trabajar. Como ya se menciono antes vamos a utilizar la función odeint para resolver el sistema de ecuaciones.

El codigo[4] que utilizamos para resolver el sistema de ecuaciones, define lo que se conoce como campo vectorial. Se escoge poner la función que define el campo vectorial en su propio modulo, pero esto no es necesario. Los argumentos de la función están configurados para ser usados con la función odeint: el tiempo t es el segundo argumento.

```python
#Definimos un campo vectorial
def vectorfield(w, t, p):
    """
    Define las ecuaciones diferenciales para un sistema masa-resorte acoplado.

    Argumentos:
        w :  vector de las variables establecidas:
                  w = [x1,y1,x2,y2]
        t :  tiempo
        p :  vector de los parámetros:
                  p = [m,k,L1,L2]
    """
    x1, y1, x2, y2 = w
    m, k , L1, L2 = p

    # Creamos f = (x1',y1',x2',y2'):
    f = [y1,
          k * ( -2*x1 + L1 - L2 + x2 ) / m,
         y2,
         - k * ( x2 - x1 - L2 )/ m]

    return f
```

Figura 2:

Este script usa odeint para resolver las ecuaciones que vienen con un set de parametros, condiciones iniciales e intervalos de tiempo.

```python
# Utilizamos ODEINT para resolver las ecuaciones diferenciales definidas por el campo vectorial.
from scipy.integrate import odeint

# Valores de los parámetros
# Masas:
m = 1.0

# Constantes de los resortes
k = 1.0

# Longitudes naturales
L1 = 1.0
L2 = 1.0

# Condiciones iniciales
# x1 y x2 son los desplazamientos iniciales
# y1 y y2 son las velocidades iniciales
x1 = 1.0
y1 = 0.0
x2 = 0.0
y2 = 0.0

#Pámetros de reslución del EDO
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 30.0
numpoints = 750
```

Figura 3:

y por último obtenemos la solución del sistema de ecuaciones utilizando la función odeint

```
# Creamos las muestras de tiempo para el resultado del solucionador de EDO
# Usamos un gran número de puntos, para hacer una gráfica de la solución con
# buen aspecto.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]


# Empacamos los parámetros y las condiciones iniciales:
p = [m, k, L1, L2]
w0 = [x1, y1, x2, y2]

# Llamamos al solucionador de EDO.
wsol = odeint(vectorfield, w0, t, args=(p,),
              atol=abserr, rtol=relerr)
```

Figura 4:

La figura 2 nos muestra el desplazamiento de las masas para este sistema acoplado, claro, una vez que obtenemos la solución al sistema de ecuaciones.

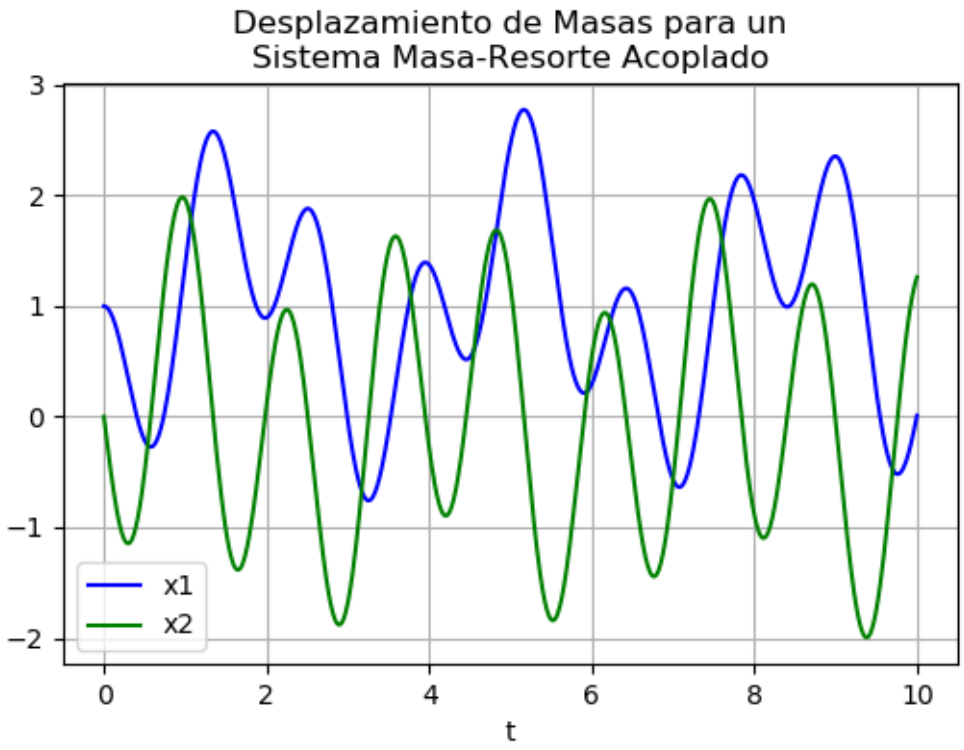

Figura 5:

Conclusiones

Las funciones de numpy y de scipy para resolver estos sistemas de ecuaciones son muy eficientes, ya que permiten que en muy pocas lineas de código se pueda dar solución a un problema que en apariencia pareciera mas complicado de resolver numericamente.

Referencias

[1] http://computacional1.pbworks.com/w/page/133186848/Actividad9

[2] https://farside.ph.utexas.edu/teaching/315/Waves/node18.html

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html

[4] https://scipy-cookbook.readthedocs.io/items/CoupledSpringMassSystem.html