Suplemento Computacional

Física de Oscilaciones y Ondas

Sebastian Bustamante Jaramillo
macsebas33@gmail.com

UNIVERSIDAD
DE ANTIOQUIA
1 8 0 3

Facultad de Ciencias Exactas y Naturales
Universidad de Antioquia

Índice general

Capítulo 1

Preliminares

1.1. Motivación

La física ha evolucionado hasta un estado actual donde la mayoría de cálculos teóricos necesarios para realizar investigación de frontera requieren de una gran componente computacional. Desde la corroboración entre teoría y experimento, la predicción y control de los resultados de un experimento hecho a posteriori y la recreación de condiciones imposibles de lograr experimentalmente, tales como simulaciones cosmológicas del universo a gran escala o complejos sistemas atómicos. Estos son sólo algunos ejemplos representativos del papel de la computación en la física moderna. Debido a esto, el principal objetivo del suplemento computacional es la introducción temprana en los cursos de física básica de herramientas computacionales que serán de utilidad a los estudiantes en este curso específico y durante el transcurso de sus carreras científicas.

1.2. Instalación de Paquetes

En la totalidad de esta guía será usado el lenguaje de programación *Python* como referente para todos las prácticas y ejercicios computaciones. La principal motivación de esto es su facilidad de implementación en comparación a otros lenguajes también de amplio en ciencia. Además es un lenguaje interpretado, lo que permite una depuración más sencilla por parte del estudiante, sin necesidad de usar más complicados sistemas de depuración en el caso de lenguajes compilados como C o Fortran. *Python* es un lenguaje de código abierto, lo que permite la libre distribución del paquete y evita el pago de costosas licencias de uso, además la gran mayoría de paquetes que extienden enormemente la funcionalidad de *Python* son también código abierto y de libre distribución y uso.

A pesar de que *Python* es un lenguaje multiplataforma, permitiendo correr scripts python en Linux, Windows y Mac, acá solo se indicará el método de instalación para distribuciones Linux basadas en Debian.

La última versión de *Python* de la rama 2 es 2.7.4 y de la rama 3 es la 3.3.1, debido a ligeras incompatibilidades entre ambas ramas de desarrollo, será utilizada la rama 2 en una de sus últimas versiones. En orden, para instalar *Python* en una versión Linux basta con descargarlo directamente de los repositorios oficiales[1], en el caso de una distro basada en Debian el gestor de paquetes es `apt-get`, y desde una terminal se tiene

```
\$ apt-get install python2.7
```

también puede descargarse directamente desde la página oficial del proyecto `http://python.org/`.

Una vez instalada la última versión de *Python* , es necesario instalar los siguiente paquetes para el correcto desarrollo de las aplicaciones del curso:

iPython

iPython es un shell que permite una interacción más interactiva con los scripts de python, permitiendo el resaltado de sintaxis desde consola, funciones de autocompletado y depuración de código más simple. Para su instalación basta descargarlo de los repositorios oficiales

```
\$ apt-get install ipython
```

o puede de descargarse de la página oficial `http://ipython.org/`. También puede encontrarse documentación completa y actualizada en esta página, se recomienda visitarla frecuentemente para tener las más recientes actualizaciones.

NumPy

NumPy es una librería que extiende las funciones matemáticas de *Python* , permitiendo el manejo de matrices y vectores. Es esencial para la programación científica en *Python* y puede ser instalada de los repositorios

```
\$ apt-get install python-numpy
```

[1] En la mayoría de distribuciones Linux *Python* viene precargado por defecto.

La última versión estable es la 1.6.2. En la página oficial del proyecto puede encontrarse versiones actualizadas y una amplia documentación `http://www.numpy.org/`.

SciPy

SciPy es una amplia biblioteca de algoritmos matemáticos para *Python* , esta incluye herramientas que van desde funciones especiales, integración, optimización, procesamiento de señales, análisis de Fourier, etc. Al igual que los anteriores paquetes, puede ser instalada desde los repositorios oficiales

```
\$ apt-get install python-scipy
```

Una completa documentación del paquete puede ser encontrada en `http://docs.scipy.org/doc/scipy/reference/`. La última versión estable es la 0.11.0 y puede ser encontrada en la página oficial del proyecto `http://www.scipy.org/`.

Matplotlib

Matplotlib es una completa librería con rutinas para la generación de gráficos a partir de datos. Aunque en su estado actual está enfocada principalmente a gráficos 2D, permite un amplio control sobre el formato de las gráficas generadas, dando una amplia versatilidad a los usuarios. Su instalación puede realizarse a partir de los repositorios oficiales

```
\$ apt-get install python-matplotlib
```

La última versión estable es la 1.2.1. y puede encontrarse en la página oficial del proyecto `http://matplotlib.org/`. Una amplia documentación está disponible en `http://matplotlib.org/1.2.0/contents.html`.

MayaVi2

MayaVi2 es una librería para la visualización científica en python, en especial para gráficos 3D, permitiendo funciones avanzadas como renderizado, manejo de texturas, etc. Se encuentra en los repositorios oficiales

```
\$ apt-get install mayavi2
```

La versión 2 es una versión mejorada de la original, estando más orientada a la reutilización de código. Por defecto incluye una interfaz gráfica que facilita su manejo. La página oficial del proyecto es `http://mayavi.sourceforge.net/`.

Tkinter

TKinter es una librería para la gestión gráfica de aplicaciones in *Python* y viene por defecto instalada, aún así puede ser instalada de los repositorios oficiales

```
\$ apt-get install python-tk
```

La página oficial del proyecto es `http://wiki.python.org/moin/TkInter`. Para el desarrollo de entornos gráficos existen otras llamativas alternativas como PyGTK o PyQt, pero debido a la facilidad de uso y a ser la librería estándar soportada, *TKinter* será usada en este curso.

Capítulo 2

Oscilaciones

El concepto de oscilación es ampliamente usado en ciencia y se aplica para cualquier cantidad que presente fluctuaciones o perturbaciones en función del tiempo, ya sean periódicas o no. En este capítulo se presentan algunos ejemplos de oscilaciones para sistemas mecánicos y electromagnéticos, que a pesar de su simplicidad, permiten ahondar en los detalles físicos de este tipo de fenómenos.

La forma estándar de abordar un problema en una disciplina científica consiste en el estudio de situaciones ideales y muy particulares para llegar luego, usando refinamientos y consideraciones posteriores, a descripciones realistas y más generales. Por este motivo se comienza con demostraciones computacionales aplicadas a sistemas ideales, como el péndulo simple, el sistema masa resorte, etc. En demostraciones siguientes se abordarán aspectos cada vez más complejos de estos sistemas.

2.1. Demostración 1: Péndulo Simple Ideal

Como primer caso se aborda el péndulo simple. Este consiste en un sistema de una masa m con dimensión despreciable y bajo la acción de la gravedad $\boldsymbol{g}$, además pende de una cuerda tensa de longitud l y sujeta en un punto fijo. A partir de la figura 2.1, la ecuación de movimiento está dada por

$$m\frac{d^2\boldsymbol{r}}{dt^2} = m\boldsymbol{g} + \boldsymbol{T} \tag{2.1}$$

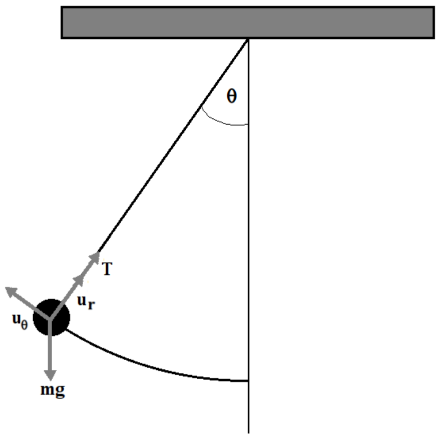

Figura 2.1: Péndulo simple bajo la acción del campo gravitacional.

En coordenadas polares se obtiene

$$ml\dot{\theta}^2 = T - mg\cos\theta \tag{2.2}$$
$$ml\ddot{\theta} = -mg\sin\theta \tag{2.3}$$

Como primera demostración computacional de este capítulo se solucionará el movimiento del péndulo simple a partir de las ecuaciones 2.2 y 2.3. Para esto se asumirá que la amplitud de oscilación del péndulo es pequeña de tal forma que $\theta \approx \sin\theta$ y $\cos\theta \approx 1$, obteniendo

$$\ddot{\theta} = -\frac{g}{l}\theta \tag{2.4}$$

Usando un ansatz de la forma $\theta(t) = e^{\lambda t}$ se llega a la solución

$$\theta(t) = \theta_0 \sin(\omega_0 t + \delta) \tag{2.5}$$

donde θ_0 y δ son la amplitud y la fase respectivamente y constituyen las condiciones iniciales. La frecuencia ω_0 está definida por

$$\omega_0 = \sqrt{\frac{g}{l}} \tag{2.6}$$

En el siguiente script de *Python* se grafica esta solución para diferentes valores de la amplitud y la fase.

```
#!/usr/bin/env python
#==========================================================
# DEMOSTRACION 1
# Grafica de soluciones aproximadas del pendulo simple
#==========================================================
import numpy as np
import matplotlib.pylab as plt

#Solucion
def Theta(t):
    theta = theta0*np.sin( omega0*t + delta )
    return theta

#Gravedad
g = 9.8
#Longitud
l = 1
#Frecuencia
omega0 = np.sqrt( g/l )
#Tiempos
tiempo = np.arange( 0, 10, 0.1 )

#SOLUCION 1
#Amplitud
theta0 = 0.05
#Fase
delta = 0.0
#Grafica
plt.plot( tiempo, Theta(tiempo), label='solucion 1' )

#SOLUCION 2
#Amplitud
theta0 = 0.05
#Fase
delta = np.pi
#Grafica
plt.plot( tiempo, Theta(tiempo), label='solucion 2' )

#SOLUCION 3
#Amplitud
theta0 = 0.1
#Fase
```

```
delta = 0.0
#Grafica
plt.plot( tiempo, Theta(tiempo), label='solucion 3' )

plt.legend()
plt.show()
```

El resultado que se obtiene es

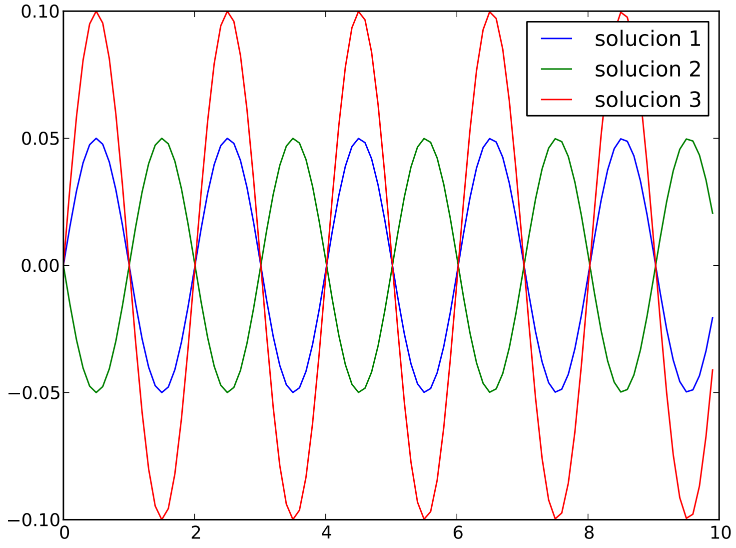

Figura 2.2: Soluciones aproximadas obtenidas para el péndulo simple.

En lo siguiente se explica cada porción de código.

```
import numpy as np
import matplotlib.pylab as plt
```

Esto corresponde al cargado de las librerías *NumPy* , bajo el alias de np y *Matplotlib* bajo el alias de plt, ambas son necesarias para el desarrollo de todo el código.

```
#Solucion
def Theta(t):
    theta = theta0*np.sin( omega0*t + delta )
    return theta
```

En este parte se define la solución obtenida en la ecuación 2.5 para cualquier tiempo dado. Las funciones matemáticas estándar, tales como exponencial, logaritmo, trigonométricas, hiperbólicas, etc. Son accedidas desde el módulo *NumPy* como `np.sin`, `np.sinh`, `np.exp`, `np.log`, etc.

```
#Gravedad
g = 9.8
#Longitud
l = 1
#Frecuencia
omega0 = np.sqrt( g/l )
#Tiempos
tiempo = np.arange( 0, 10, 0.1 )
```

Se define la gravedad, la longitud de la cuerda (en metros), la frecuencia de oscilación y finalmente, usando la función `arange` de la librería *NumPy* , se construye un arreglo de tiempos donde es evaluada la solución, de 0 a 10 segundos con saltos de 0.1.

```
#SOLUCION 1
#Amplitud
theta0 = 0.05
#Fase
delta = 0.0
#Grafica
plt.plot( tiempo, Theta(tiempo), label='solucion 1' )
```

La primera solución es obtenida para una amplitud de $\theta_0 = 0{,}05$ radianes y una fase $\delta = 0$. La última línea corresponde a la construcción de la gráfica de la solución. Para esto se usa la función `plot` de la librería *Matplotlib* . El primer argumento corresponde a los datos asociados al eje x, en este caso `tiempo`, mientras el segundo argumento son los datos asociados al eje y, en este caso la solución evaluada en el tiempo, es decir `Theta(tiempo)`. El argumento `label` indica el nombre que tendrá la solución en la gráfica final, esto se denomina etiqueta de la función.

```
plt.legend()
plt.show()
```

Finalmente se termina el script con la función `legend`, la cual muestra en pantalla las etiquetas puestas a cada gráfica. Y la función `show` que muestra en pantalla todas las soluciones.

2.2. Demostración 2: Solución Exacta del Péndulo

La solución propuesta en la demostración anterior está basada en la aproximación de pequeñas oscilaciones, para la cual $\sin\theta \approx \theta$ y $\cos\theta \approx 1$, aún así, a medida que el movimiento sea de mayor amplitud, esta aproximación deja de ser válida y se la solución general debe obtenerse directamente de las ecuaciones 2.2 y 2.3

Una forma conveniente de reescribir las ecuaciones de movimiento es derivando 2.2 respecto al tiempo e introduciendo la ecuación 2.3

$$\begin{aligned}\dot{T} &= \dot{\theta}\left(2ml\ddot{\theta} - mg\sin\theta\right)\\ \dot{T} &= -\dot{\theta}\left(2mg\sin\theta + mg\sin\theta\right)\\ \dot{T} &= -3\dot{\theta}mg\sin\theta \qquad (2.7)\end{aligned}$$

Definiendo la velocidad angular $\omega = \dot{\theta}$, el sistema de ecuaciones originales junto con la ecuación 2.7 queda

$$\dot{\theta} = \omega \qquad (2.8)$$

$$\dot{\omega} = -\frac{g}{l}\sin\theta \qquad (2.9)$$

$$\dot{T} = -3\omega mg\sin\theta \qquad (2.10)$$

Esta forma se denomina sistema de ecuaciones diferenciales lineales y es esencial para la solución exacta a partir de algoritmos de integración numérica.

Las condiciones iniciales para la solución aproximada son completamente determinadas a partir de la fase δ y la amplitud θ_0. En el caso de la solución exacta es necesario suministrar tres condiciones para cada una de las variables de las ecuaciones 2.8 - 2.10, es decir, la posición angular inicial del péndulo θ_{t_0}, la velocidad angular inicial ω_{t_0} y la tensión inicial T_{t_0}. Aún así, la tensión inicial depende de las otras dos condiciones a través de la ecuación 2.2

$$T_{t_0} = ml\omega_{t_0}^2 + mg\cos\theta_{t_0} \qquad (2.11)$$

En esta demostración se calcularán ambas soluciones para un péndulo con condiciones iniciales fijadas. La solución numérica se realiza a partir del la rutina de integración numérica `odeint` del módulo `integrate` del paquete *SciPy* .

```
#!/usr/bin/env python
#=============================================================
# DEMOSTRACION 2
# Comparacion de solucion completa y aproximada para el
# pendulo simple
#=============================================================
import numpy as np
import matplotlib.pylab as plt
import scipy.integrate as integ

#Solucion aproximada
def Theta(t):
    theta = theta0*np.sin( omega0*t + delta )
    return theta

#Ecuaciones de movimiento
def dF(Y, t):
    #Valor anterior de theta
    theta = Y[0]
    #Valor anterior de omega
    omega = Y[1]
    #Valor anterior de la tension
    tension = Y[2]
    #Derivada de theta
    Dtheta = omega
    #Derivada de omega
    Domega = -g*np.sin( theta )/l
    #Derivada de la tension
    Dtension = -3*omega**m*g*np.sin( theta )
    return (Dtheta, Domega, Dtension)

#Gravedad
g = 9.8
#Longitud
l = 1
#Masa
m = 1
#Frecuencia
omega0 = np.sqrt( g/l )
#Tiempos
tiempo = np.arange( 0, 10, 0.01 )

```

```
#SOLUCION APROXIMADA
#Amplitud
theta0 = np.pi/4
#Fase
delta = np.pi/2.
#Grafica
plt.plot(tiempo, Theta(tiempo), label=’solucion aproximada’)

#SOLUCION NUMERICA
#Angulo inicial
theta_t0 = np.pi/4
#Velocidad angular inicial
omega_t0 = 0.0
#Tension inicial
tension_t0 = m*l*omega_t0**2 + m*g*np.cos( theta_t0 )
#Condiciones iniciales
cond_ini = ( theta_t0, omega_t0, tension_t0 )
#Solucion numerica
theta_t, omega_t, tension_t = np.transpose(
integ.odeint( dF, cond_ini, tiempo ) )
#Grafica
plt.plot( tiempo, theta_t, label=’solucion numerica’ )

plt.legend()
plt.show()
```

Para ambas soluciones se ha asumido una amplitud de $\theta_0 = 45^o = \pi/4$ y las condiciones iniciales son definidas de tal forma que el péndulo sea soltado desde su máxima amplitud. En el caso de la solución aproximada, esto implica una amplitud $\theta_0 = \pi/4$ y una fase de $\delta = \pi/2$. Para la solución aproximada, las condiciones equivalentes son $\theta_{t_0} = \pi/4$ y $\omega_{t_0} = 0$. El resultado de la integración para 10 segundos es mostrado en la figura 2.3.

Cambiando la amplitud inicial a un valor $\theta = 10^o \approx 0{,}17$ ambas soluciones son muy similares, indicando el rango de la validez de la aproximación inicial. El resultado para 10 segundos es mostrado en la figura 2.4.

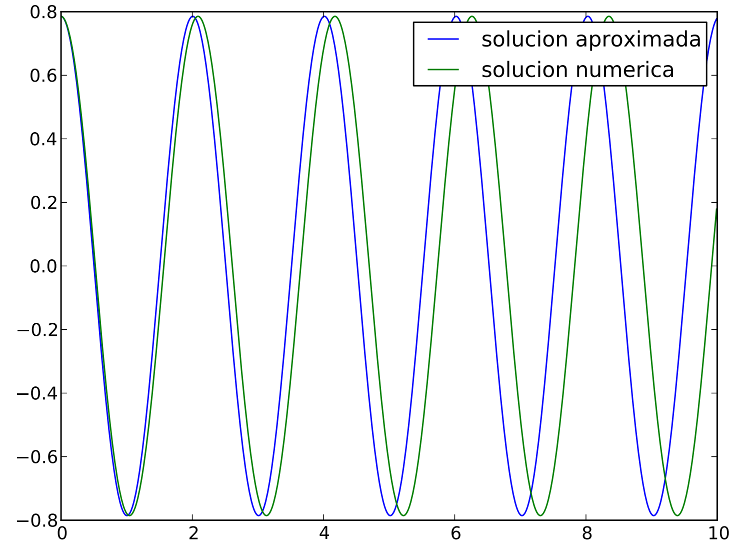

Figura 2.3: Comparación entre la solución exacta numérica y la solución aproximada. Debido a la mayor amplitud de oscilación, la aproximación de pequeñas oscilaciones deja de ser válida y difiere considerablemente de la solución real.

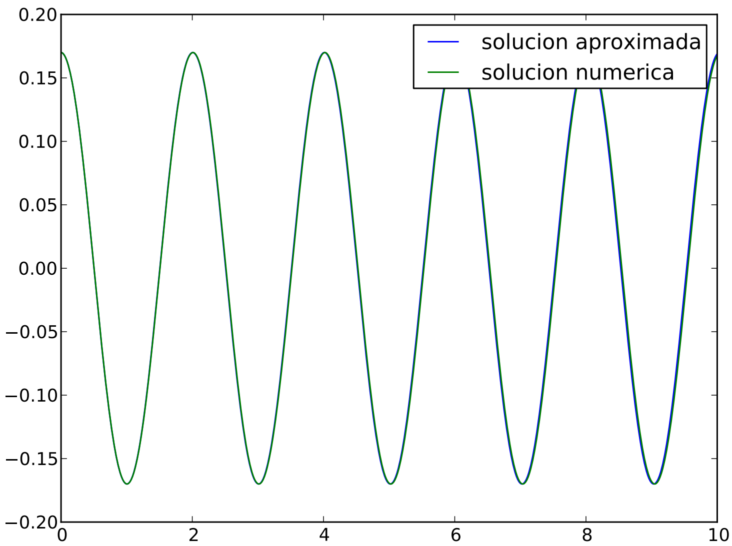

Figura 2.4: Comparación entre la solución exacta numérica y la solución aproximada. Para pequeñas amplitudes ambas soluciones son muy similares.

A continuación se explican las nuevas partes introducidas en el código en comparación con la demostración 1.

```
import scipy.integrate as integ
```

Esto corresponde al cargado del módulo `integrate` de *SciPy* bajo el alias de `integ` para las funcionalidades de integración numérica.

```
#Ecuaciones de movimiento
def dF(Y, t):
    #Valor anterior de theta
    theta = Y[0]
    #Valor anterior de omega
    omega = Y[1]
    #Valor anterior de la tension
    tension = Y[2]
    #Derivada de theta
    Dtheta = omega
    #Derivada de omega
    Domega = -g*np.sin( theta )/l
    #Derivada de la tension
    Dtension = -3*omega**m*g*np.sin( theta )
    return (Dtheta, Domega, Dtension)
```

Se define la función que contiene las derivadas de cada una de las variables del sistema, acorde al sistema 2.8 - 2.10. `Y` es un arreglo que contiene las variables θ, ω y T en el tiempo anterior respectivamente y `t` el tiempo actual.

```
#SOLUCION NUMERICA
#Angulo inicial
theta_t0 = np.pi/4
#Velocidad angular inicial
omega_t0 = 0.0
#Tension inicial
tension_t0 = m*l*omega_t0**2 + m*g*np.cos( theta_t0 )
#Condiciones iniciales
cond_ini = ( theta_t0, omega_t0, tension_t0 )
#Solucion numerica
theta_t, omega_t, tension_t = np.transpose(
integ.odeint( dF, cond_ini, tiempo ) )
#Grafica
plt.plot( tiempo, theta_t, label='solucion numerica' )
```

Finalmente se realiza la integración de la solución exacta del péndulo simple. Se dan las condiciones iniciales para el ángulo y la velocidad angular inicial, para la tensión se usa la ecuación 2.11. Luego, en un arreglo `cond_ini` se ponen las tres condiciones para luego llamar la función `integ.odeint`. Esta tiene como primer argumento la función `dF` con las ecuaciones de movimiento del sistema, segundo argumento el arreglo de condiciones iniciales y como tercero el arreglo de tiempo donde se desea evaluar la solución. Finalmente se grafica la solución.

Capítulo 3

Bibliografía