Informe Tarea 2: Métodos Numéricos

Ignacio Andrés Sánchez Barraza

October 1, 2015

1 Pregunta 1

1.1 Introducción

- Se tiene una partícula de masa m que se mueve verticalmente sólo en el eje Y rebotando contra un suelo que oscila sinusoidalmente con amplitud A y frecuencia ω. El choque contra el suelo es inelástico siguiendo la siguiente regla de choque:

$$v'_p(t^*) = (1+\eta)v_s(t^*) - \eta v_p(t^*)$$

donde t^* es el instante del bote o choque contra el suelo, v_p y v'_p son las velocidades justo antes y justo despues del bote, v_s es la velocidad del suelo en ese instante y η es un coeficiente de restitución (η entre 0 y 1; η=1 corresponde al caso elástico).

Inicialmente la partícula está en contacto con el suelo y con velocidad hacia arriba mayor que la velocidad de éste. Se desea entonces programar una rutina en python que pueda entregar la velocidad y altura de la masa justo despúes del choque contra el suelo $v'_{p_{n+1}}$ e $y_{p_{n+1}}$, teniendo como entrada los datos v_{p_n} e y_{p_n}, la velocidad y altura de la masa justo antes del choque.

1.2 Procedimiento y resultados

- Como primer paso para lograr esto, se debió idear una función como la sugerida en la ayuda de la pregunta. Esta función encontraría los ceros o raíces de una función a través del módulo *scipy.optimize.bisect* (que toma como argumentos una funcióm y dos puntos ,en los cuales la función debe tener signos distintos). Para ello se diseñó un método por el cual ir evaluando la función en valores que se variarían hasta que la función cambiara de signo, con ello se obtuvo:

```
from scipy.optimize import bisect as bi
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import numpy as np
def raiz(f,h):
    contador=0
    while f(contador)>=0:
        contador+=h
    return bi(f,contador-h,contador)
```

Como se muestra en el código, se define la función $raiz(f,h)$, que busca la raíz de f con un paso h, iterando hasta encontrar donde cambia de signo y retorna la raíz de f por el método de bisección. Ahora bien en este caso específicamente, la función a la cual se deben encontrar las raíces, es la función $g(t) = y_p(t) - y_s(t)$, ie., la diferencia entre la altura del suelo y de la masa. Por ello se asumió que la función evaluada en valores desde el origen hacia el infinito era positiva y se detectaría cuando la función se hacia negativa, ie, la altura de la masa era menor a la del suelo (esto debido a que como la masa está restringida a moverse en el eje Y, sigue una trayectoria de lanzamiento vertical hacia arriba descrita por la ecuación $y_p(t) = y_o + v_o t - \frac{gt^2}{2}$, por lo que se tiene una trayectoria de tipo parabólica con respecto al tiempo, con concavidad negativa, por lo que la suposición es acertada). También a partir de la posición de la masa se pudo obtener su velocidad $v_p = v_o t - gt$ y la del suelo,suponiendo un movimiento sinusoidal del tipo $y_s(t) = A\sin(\omega t + fase)$, $v_s(t) = A\omega\cos(\omega t + fase)$ (con la fase la diferencia de altura debido a $y_p(t_{choque})$ que hay que

considerar si se quiere repetir el movimiento de la masa después de cada choque, suponiendo que la pelota siempre parte desde el suelo).
Hecho esto, cabe destacar que como el algoritmo que se usó para encontrar los puntos con los cuales el método de la bisección puede trabajar depende del paso que se le otorga a la variación del argumento de la misma (para asi encontrar cuando se hacia negativa dicha función), se debió utilizar un paso "pequeño", es decir, que mientras mas pequeño fuera ese paso, mejor era la aproximación de la raíz y menor el tiempo de ejecución del algoritmo para encontrarlas. Por ello también es que se prefirió definir la función en python $raiz(f, h)$ dejando como argumento la función f a la cual encontrar las raíces y el paso h descrito anteriormente explícito para poder manipular la precisión del método(otra idea sería utilizar otros métodos para encontrar raíces para optimizar el tiempo de ejecución, con ordenes del mismo menores al método de bisección como el método de Newwton y otros, pero que no garantizan encontrar las raíces como si lo hace el método utilizado).
Dicho esto y definidas entonces la velocidad del suelo y la masa así como sus alturas, y teniendo la relación entre la velocidad justo antes y despúes del choque (ie, $v_p(t_{choque})$ y $v'_p(t_{choque})$), se procedió a implementar el programa con dichas funciones (dependientes de las constantes del problema η, ω, g, A, etc.) y que retornaran los valores de la altura y velocidad de la masa despúes del rebote, es decir. $v'_{p_{n+1}}$ e $y_{p_{n+1}}$, obteniéndose un código como éste:

```
def altyvelocidad(omega,eta,yi,vi):
    g=1
    A=1
    y0=yi
    v0=vi
    yp=lambda t:y0+v0*t-g*(t**2)/2.0
    vp=lambda t:v0-g*t
    ys=lambda t:A*np.sin(omega*t+np.arcsin(y0))
    vs=lambda t:A*omega*np.cos(omega*t+np.arcsin(y0))
    vpantes=lambda t:vp(t)
    vpdespues=lambda t:(1+eta)*vs(t)-eta*vpantes(t)
    fraiz=lambda t:y0+v0*t-g*(t**2)/2.0 - A*np.sin(omega*t+np.arcsin(y0))
    traiz=raiz(fraiz,0.001)
    altp=yp(traiz)
    velpdespues=vpdespues(traiz)
    return altp,velpdespues
```

2 Pregunta 2

2.1 Introducción

- Debido a que el sistema descrito tiende a generar soluciones estables (periódicas) luego de un período de relajación, se pide encontrar el N_{relax} en el cual esto ocurre, para un valor de $\eta = 0.15$ y $\omega = 1.66$.

2.2 Procedimiento y resultados

- Como ya se poseen las funciones necesarias en el algoritmo creado, basta solo evaluar las funciones en los datos entregados y buscar dichos rebotes N-ésimos en los cuales el sistema queda en "relajación". Para ello se utilizará la sugerencia del enunciado que propone graficar la velocidad de la masa con el número de rebotes y chequear el valor de N que cumple con dejar el estado relajado. Para ello se crea un código que grafique un arreglo para el número rebotes definido con el módulo *numpy.linspace* y otro para las velocidades justo después de cada rebote, es decir:

```
#eta=0.15
#omega=1.66
fig=plt.figure(1)
fig.clf
fig.set_size_inches(16, 8, forward=True)
n=np.linspace(0,20,num=21)
y0=0
v0=10
vpn=[]
for i in n:
```

```
    yi=y0
    vi=v0
    vpn.append(vi)
    y0=(altyvelocidad(1.66,0.15,yi,vi))[0]
    v0=(altyvelocidad(1.66,0.15,yi,vi))[1]
plt.plot(n,vpn,'r-')
```

y con el cual se obtiene el siguiente gráfico para 20 rebotes:

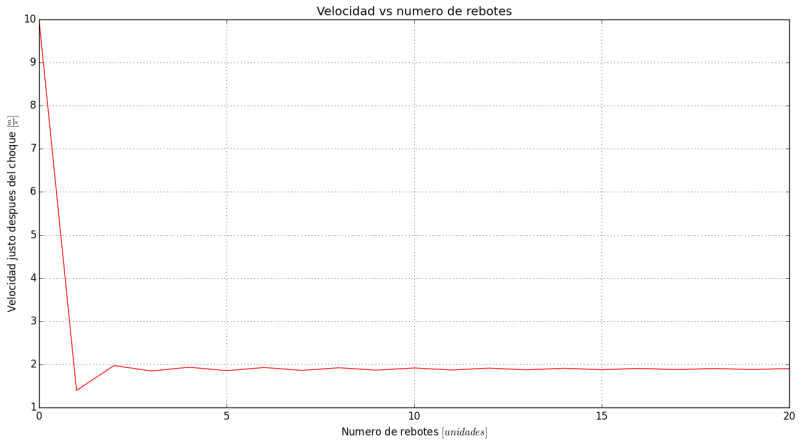

Figure 1: Gráfico Velocidad de la masa (o pelota) vs Número de rebotes.

2.3 Conclusiones

- Como se ve en el gráfico el valor de N que estabiliza o relaja el sistema es $N = 2$, dado que desde ese rebote la velocidad de la masa se mantiene variando en una vecindad pequeña alrededor de $V'_{p_n} = 2$, lo que se interpreta como que la masa se encuentra pegada al suelo y se mueve solidariamente con éste, entonces $N_{relax} = 2$.

3 Pregunta 3

3.1 Introducción

- En esta pregunta se pide graficar una vez más para dos valores de ω entre 1.66 y 1.7 para comparar los N_{relax} de ambos casos.

3.2 Procedimiento y resutados

- Para ello usamos un código análogo al de la pregunta 2 cambiando los valores para ω y graficando los valores de las velocidades para ambos omegas en una misma figura para asi poder compararlos, obteniendo un código como el siguiente:

```
#omega=1.68 y 1.7
#eta=0.15
n=np.linspace(0,50,num=51)
vpna=[]
vpnb=[]
y0a=0
v0a=10
y0b=0
v0b=10
for i in n:
    yia=y0a
    via=v0a
    yib=y0b
```

```
    vib=v0b
    vpna.append(via)
    vpnb.append(vib)
    y0a=(altyvelocidad(1.68,0.15,yia,via))[0]
    v0a=(altyvelocidad(1.68,0.15,yia,via))[1]
    y0b=(altyvelocidad(1.7,0.15,yib,vib))[0]
    v0b=(altyvelocidad(1.7,0.15,yib,vib))[1]
fig=plt.figure(1)
fig.clf
fig.set_size_inches(16, 8, forward=True)
plt.plot(n,vpna,'r-')#Primer grafico
plt.plot(n,vpnb,'b-')#Segundo grafico
```

y con ello se obtiene el siguiente gráfico comparando los valores para $\omega = 1.68$ y $\omega = 1.7$:

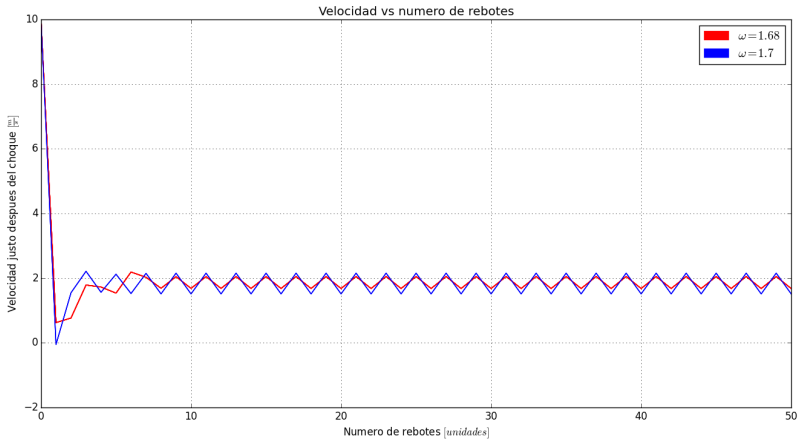

Figure 2: Gráfico Velocidad de la masa (o pelota) vs Número de rebotes.

3.3 Conclusiones

- Es fácil notar que los N_{relax} para ambos valores de ω son comparables a simple vista, y que el N_{relax} para $\omega = 1.68$, ie., $N = 7$, relaja o estabiliza al sistema, mientras que el sistema con $\omega = 1.7$ se estabiiza para $N = 2$. Aún así se ve que una pequeña variación en la frecuencia con que se mueve el suelo, afecta en una cantidad no despreciable la cantidad de rebotes para que el sistema se relaje.

4 Pregunta 4

4.1 Introducción

- En esta pregunta se pide seguir utilizando $\eta = 0.15$ y $\omega = 1.66, ..., 1.79$ y graficar para cada valor de ω los valores de las velocidades para rebotes $n = 2N_{relax}, ..., 2N_{relax} + 49$, es decir, 50 valores de v'_n por cada valor de ω.

4.2 Procedimiento y resultados

- Para poder abordar este problema, se usó un código análogo a las preguntas 2 y 3, pero que permitiese graficar velocidad vs frecuencia angular, graficando 50 puntos (velocidad) para cada valor de frecuencia. Para esto se debían almacenar los valores de las velocidad justo después de cada rebote definido anteriormente por la lista n, por lo que se definió una lista *omegas* que contuviera la lista de los valores de ω y otra lista de listas v_{p_n} que contuviese las listas de valores de las velocidades asociadas a cada valor de ω:

```
nrelax=2
n=[]
```

```
for j in range(50):
    n.append(2*nrelax+j)
y0=0
v0=10
numomegas=20
omegas=np.linspace(1.66,1.79,num=numomegas)
vpn=np.zeros((numomegas,50))
rebote=0
contomega=0
for h in omegas:
    j=0
    for i in n:
        yi=y0
        vi=v0
        while rebote<=i:
            y0=(altyvelocidad(h,0.15,yi,vi))[0]
            v0=(altyvelocidad(h,0.15,yi,vi))[1]
            rebote+=1
        vpn[contomega][j]=v0
        j+=1
    contomega+=1
```

Hecho esto, para poder graficar ambas listas con dimensiones distintas, se procedió a graficar tomando los valores de ω y los primeros valores de cada sublista dentro de la lista v_{p_n}, luego lo mismo para los segundos valores, etc.

```
for i in range(50):
    plt.plot(omegas,vpn[:,i],'r-')
```

Y con esto se obtuvo el siguiente gráfico:

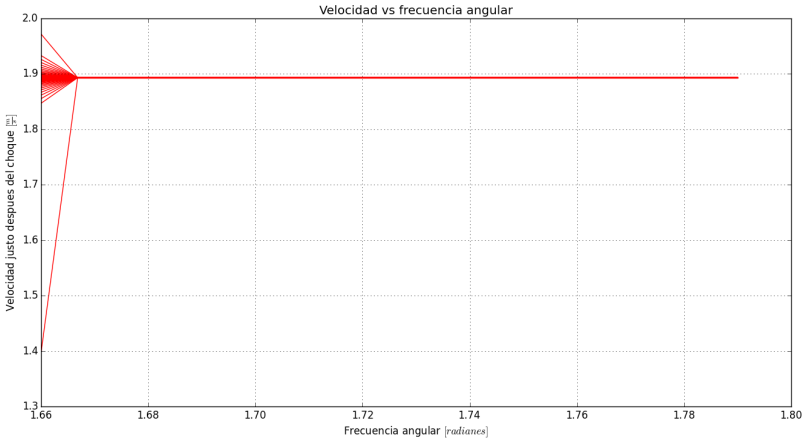

Figure 3: Gráfico Velocidad de la masa (o pelota) vs Frecuencia angular.

4.3 Conclusiones

- Conluyendo sobre este último resultado, se ve que para los distintos valores de ω las velocidades de la masa convergen a un valor que se mantiene constante luego de un valor fijo de ω en este caso, convergen a $v_p \approx 1.9$ a partir de un $\omega \approx 1.665$