Geoscience: Applied Geophysics., Edu(1):1

VPYTHON

Aplicaciones a la Física Educativa

M.C. Omar Mireles

August,2012

Abstract

Se presenta en forma breve una descripción generalizada de las aplicaciones de la paqueteria visual python en la física computacional básica. Este trabajo se crea con la intención de que sea un manual de estudio. Los ejercicios comienzan a subir el nivel de dificultad para el usuario, sin embargo al final del manual se anexan los códigos completos, así como también se encuentran en la pagina www.geoscience.com.mx, en la sección de servicios/software/python. Muchos de los ejemplos aquí explicados tienen su origen en la red o en proyectos especializados en el tema y propiedad de otros autores (la mayoria señalados en la bibliografía).

1 Instalación

Una reseña completa acerca de la instalación de visual python se puede encontrar en García (2008), así como algunos ejemplos de su uso en García (2008), Sanders (2010) y Marzal & Gracia (2003).

En lo personal me funciono escribir en la terminal los siguientes códigos

- sudo apt-get install python-visual
- sudo apt-get install libgtkglextmm-x11-1.2-dev
- sudo apt-get install python-cairo-dev

- sudo apt-get install python-gobject-dev
- sudo apt-get install python-gtk2-dev
- sudo apt-get install glade

Para saber si esta bien instalado visual python (de ahora en adelante vpython), abre una terminal, escribe en ella python. Al hacer esto te mandara al ambiente python. Ahora escribe

>>> from visual import sphere
>>> s = sphere()

Si al hacer esto se genera una esfera tridimensional, todo esta bien instalado (figura 1). En caso contrario vuelve a intentarlo.

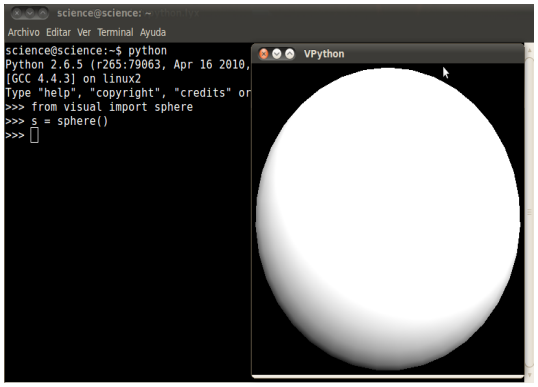

Figura 1. Vista de los comandos desde el shell de python y la esfera que se genera al comprobar que visual pyhton este funcionando correctamente.

2 El primer ejemplo: Gráficador

Para romper el hielo y demostrar que en general vpython es un lenguaje muy amigable construiremos un script que nos genere dos gráficas en el mismo plano cartesiano y donde cada una de las líneas que grafiquemos este de diferente color.

Como se menciono anteriormente, la primera línea que debemos de escribir (solo en ambiente Linux) es:

```
#! /usr/bin/env python
```

Esta primera línea se escribe para poder redireccionar nuestro programa desde la Terminal hacia el interprete de Python.

```
from visual import *
from visual.graph import *
```

En estas líneas estamos importando las paqueterias generales de vpython y una especializada en gráficos.

```
ecu1 = raw_input('Escribir su ecuación usando lenguaje python: ')
ecu2 = raw_input('Escriba segunda ecuación: ')
```

El comando *raw_input* lo utilizamos para que aparesca en pantalla el texto escrito entre comillas, y despues de que el usuario introduzca la peticion, esta se guarde en una variable asignada (en nuestro caso ecu1 y ecu2). Para saber más a fondo como escribir ecuaciones en python recomiendo el tutorial "Introducción a la Programación con Python" de Marzal.

```
f1 = gcurve(color=color.cyan)
f2 = gcurve(color=color.yellow)
```

El comando gcurve tiene 4 elementos (x,y,radio,color). En este caso solo le estamos asignando un color a cada curva, los otros valores los designaremos con la siguiente orden:

```
for x in range(0,20,0.1):
    f1.plot(pos=(x,eval(ecu1)))
    f2.plot(pos=(x,eval(ecu2)))
```

Aquí, en el ciclo for se definen $(x, y, radio)$. $f1.plot$ genera la gráfica de la curva $f1$ y $f2.plot$ la de $f2$.

La salida del programa ($./grafica.py$) es un entorno gráfico (figura 2) que nos permite dos gráficas conjuntas.

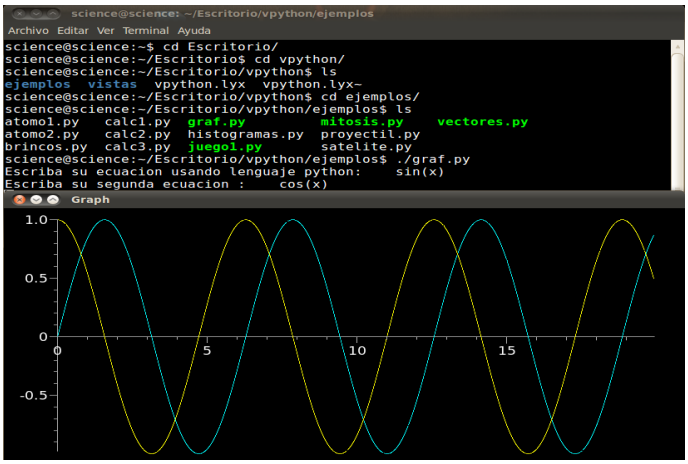

Figura 2. Salida del script anterior con la función sin(x) y cos(x).

3 Átomo

Siguiendo la línea de aprendizaje, ahora crearemos una pequeña animación con más fines didácticos que físico-computacionales.

La intención de este script es ilustrar la lógica de vpython para crear objetos y dotarlos de movimiento. Vamos a crear un átomo con un núcleo y un par de electrones girando alrededor de esté. Como siempre, en linux nuestra primera línea debe ser:

```
#! /usr/bin/env python
```

La segunda línea que debe de acompañar a un script en python es la que señala las paqueterias que se van a bajar para que corra el resto del código, que en nuestro caso sería:

```
from visual import *
```

Despues debemos de crear la escena donde se van a formar todas las figuras:

```
scene2 = display(title='ATOMO: Helio',x=0, y=0,
width=600, height=500,center=(5,0,0), background=(0.6,0.3,1))
scene2.lights = [vector(10,0,0)]
```

De estas dos líneas podemos concluir que:

- *title* crea el titulo en la ventana principal.
- x y y, son la posición inicial de nuestra escena.
- *width* y *height*, son el ancho y largo de la escena.
- *center* marca la posición a partir de donde se van a crear las figuras.
- *background* crea el color de fondo de la ventana.

La segunda línea afecta a *scene*2 y se trata de un vector el cuál le coloca sombra a los objetos que formaran al átomo.

```
n1 = sphere(pos=(1,0,0), radius=1, color = color.green)
n2= sphere(pos=(2,1,0), radius=1, color = color.green)
p1 = sphere(pos=(2,0,0), radius=1, color = color.red)
p2 = sphere(pos=(1,1,0), radius=1, color = color.red)
```

Estas cuatro líneas crean cuatro esferas (2 verdes y 2 rojas), las cuales van a formar el núcleo de nuestro átomo.

```
e1 = sphere(pos=(15,7,0), radius=0.5, color = color.blue)
e2 = sphere(pos=(-3,15,5), radius=0.5, color = color.blue)
```

Con estas líneas se forman los electrones en color azul.

Hasta aquí ya tenemos terminada la escena general con la que vamos a trabajar, sin embargo todavia no le damos movimiento ni dinamicidad a nuestro trabajo. Para comenzar escribamos las siguientes tres líneas:

```
dt=0.1
curva1= curve()
curva2= curve()
```

El *dt* es el paso diferencial que vamos a utilizar. Despues creamos dos elementos abiertos (curva1 y curva2), eso significa que existen pero los cuales no tienen definidos ningun valor. La intencion de esto ultimo es crear la figura pero adaptar su existencia a nuestra necesidad. Una vez mas este ultimo paso es mas por estetica que por cualquier otra cosa. El objetivo es que las orbitas de los electrones se vallan creando al avanzar estos.

Por ultimo, vamos a dotar de movimiento a nuestros electrones:

while 1:
rate(20)
dt=0.1
e1.rotate(angle=-dt, origin=(0,0,0), axis=(-0.3,0.8,0.3))
e2.rotate(angle=-dt, origin=(0,0,0), axis=(1,0.4,0.6))
curva1.append(pos=e1.pos)
curva2.append(pos=e2.pos)

El movimiento aquí se crea con un ciclo *while* el cual corre desde 1 hasta *rate*, el cual se puede traducir en la velocidad con la que correran los electrones (entre rate sea mas chico el electron viajara mas lento). Despues le aplicamos una función llamada *rotate* a los electrones $e1$ y $e2$ y la salida de este movimiento lo asociamos a las curvas 1 y 2, las cuales se formaran al avanzar el electrón. El resultado final (*./atomo1.py*) se puede observar en la figura 3.

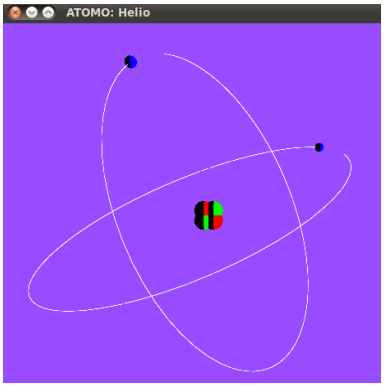

Figura 3: Imágen del átomo de Helio

Este tipo de trabajos, más que una simulación numérica presentan una animación sin mucho trasfondo matemático o físico. Pasemos ahora a algo un poco más complicado.

4 Movimiento Parabólico

Este ejemplo fue tomado casi en su totalidad del tutorial de Proyecto Physthones (García, 2008) y esta adicionado con unas pequeñas modificaciones personales para hacerlo más completo y sacarle todo el jugo posible a este script.

Antes de meternos a escribir el código recordemos un poco las leyes físicas alrededor del movimiento parabólico.

4.1 Movimiento de Proyectiles

Un proyectil es cualquier cuerpo que recibe una velocidad inicial y luego sigue una trayectoria determinada totalmente por los efectos de la aceleración gravitacional y la resistencia del aire.

Para analizar este tipo de movimiento, partiremos de un modelo idealizado que representa al proyectil como una partícula con una aceleración (debida a la gravedad) constante en magnitud y dirección. Ignoraremos los efectos del aire y la curvatura y rotación de la Tierra.

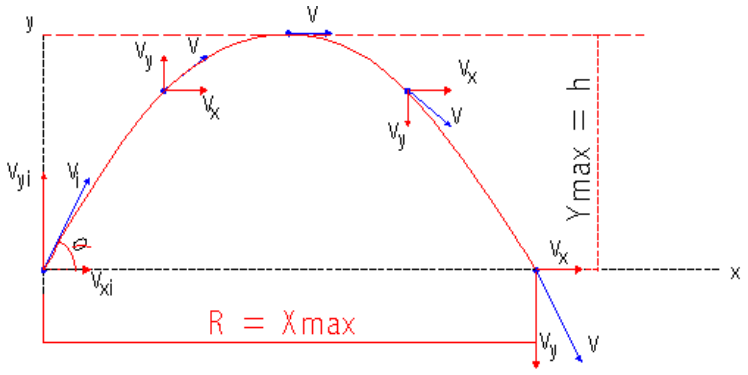

Figura 4. Diagrama vectorial del movimiento parabólico

En la figura 4 se puede observar que el movimiento de un proyectil está limitado a un plano determinado por la dirección de la velocidad inicial. La razón es que la aceleración debida a la gravedad es puramente vertical; la gravedad no puede mover un proyectil lateralmente. Por tanto, este movimiento es bidimensional. Llamaremos al plano de movimiento "*plano xy*", con x en el eje horizontal y el y vertical hacia arriba.

El análisis del movimiento de proyectiles permite tratar a x y a y por separado. Otro elemento que va a facilitar nuestro trabajo es cuando nos damos cuenta que la componente x de la aceleración es 0, y la componente y es constante e igual a $-g$. Así podemos analizar el movimiento de un proyectil como una combinación de un movimiento horizontal con velocidad constante y movimiento vertical con aceleración constante (Sears *et al*, 1999).

Para el movimiento en x tenemos que:

$$v_x = v_{0x} \tag{1}$$

$$x = x_0 + v_{0x}t \tag{2}$$

Para el movimiento en y tenemos que:

$$v_y = v_{0y} - gt \tag{3}$$

$$y = y_0 + v_{0y}t - \frac{1}{2}gt^2 \tag{4}$$

Para ver con más detalle como se obtuvieron estas expresiones puede consultar el libro de Física Universitaria, capitulo 3, sección 4.

Generalmente, lo más sencillo es tomar la posición inicial (en $t = 0$) como origen; así, $x_0 = y_0 = 0$. La figura 4 muestra la trayectoria de un proyectil que parte de (o pasa por) el origen en $t = 0$. La posición, velocidad, componentes de la velocidad y la aceleración se muestran en una serie de instantes equiespaciados. La componente x de la aceleración es 0, así que v_x es constante. La componente y es constante pero no 0, así que v_y cambia en cantidades iguales a intervalos iguales. En el punto más alto, $v_y = 0$ (Feynman *et al*, 1998).

4.2 El Script: Movimiento de un Proyectil

Iniciamos nuestro archivo como siempre:

```
#! /usr/bin/env python
from visual import *
```

Después generamos la ventana donde se visualizarán los gráficos llamándola scene y le daremos como atributos un tíitulo y le decimos que no queremos que haga autoscale. El autoscale lo que hace es mover el zoom de la cámara para que todos los objetos se vean. Suele resultar bastante incómodo y el botón central del ratón nos permitirá hacer eso manualmente sobre la marcha.

```
scene = display()
scene.title = 'Movimiento de un Proyectil'
scene.autoscale = 0
```

Ahora vamos a definir unos valores iniciales para el problema, que serán la posición y velocidad de la partícula en el instante inicial.

```
pos = vector(-10,0,0)
vel = vector(10,10,0)
```

El vector pos y vel estan definidos en un espacio tridimensional (aunque señalamos anteriormente que el movimento parabólico de un proyectil se explica en un plano bidimensional) por motivos de que vamos a crear una escena tridimensional. Ahora viene el momento en que animamos la escena. Hemos puesto un paso dt muy pequeño, cuanto más pequeño más lenta (y más precisa) será la simulación, de modo que sólo cuando t $\rightarrow$ 0 la solución coincidirá exactamente con la trayectoria analítica. Despues definimos la aceleración (componente $x = 0$ y $y = -g$) y la aceleración gravitacional (como 10) y el valor inicial cuando $t = 0$.

```
grav = 10
acel = vector(0,-grav,0)

dt=1./300.

t=0
```

Ahora crearemos los acuatro elementos principales en nuestra escena (el piso, un cañon, una bala y la trayectoria de desplazamiento de la bala):

```
proyectil = sphere(pos=pos,color=color.blue,radius=0.5)
suelo = box(pos = (0,-1,0),size = (25,0.1,25),color = color.red)
cannon = cylinder(pos = pos,axis = (1,1,0))
trayectoria = curve(color = color.white)
```

Para hacer más gráfico este modelo coloquemos tres vectores que acompañen a la bala en su trayectoria; un vector verde que representa la componente de la velocidad en x, un vector cyan que representa la componente de la velocidad en y y un vector rojo que representa la magnitud total de la velocidad.

```
velocidadtotal = arrow(color = color.red,pos = proyectil.pos,axis = vel/3.)
velocidadx = arrow(color = color.green,pos = proyectil.pos,axis = (vel.x/3.,0,0))
velocidady = arrow(color = color.cyan,pos = proyectil.pos,axis = (0,vel.y/3.,0))
```

Aquí el comando *arrow* crea un vector que depende de tres elementos; a) color, b) posición (que en nuestro caso estamos usando la misma que la del proyectil) y c) eje (en nuestro caso estamos utilizando la misma que la inicial del vector vel).

El núcleo de la animación es un bucle while que viene a decir que mientras la coordenada y del proyectil sea mayor o igual que 0 el sistema repita los pasos de dentro del bucle. Esos pasos consisten sencillamente en avanzar intervalos diferenciales y actualizar a cada paso los valores de la posición, velocidad y aceleración dados por el sistema de ecuaciones descrito anteriormente.

Según lo hemos escrito, en un ordenador lento el movimiento se producirá muy despacio y en un ordenador rápido se producirá deprisa. Para poder controlar el tiempo de manera realista debemos jugar con el número de fotogramas por segundo y el intervalo diferencial. Para eso se utiliza el comando *rate*. Si queremos, por ejemplo, 25 fotogramas por segundo (aproximadamente los que ofrece una pantalla de televisión) pondremos: *rate*(25). Finalmente esta parte del código queda como:

```
while pos.y >= 0:
    vel = vel+acel*dt
    pos = pos+vel*dt
    trayectoria.append(pos)
    proyectil.pos = pos
    velocidadtotal.pos = pos
    velocidadx.pos = pos
    velocidady.pos = pos
    velocidadtotal.axis = vel/3.
    velocidadx.axis = vector(vel.x/3.,0,0)
    velocidady.axis = vector(0,vel.y/3.,0)
    t = t+dt
    rate(50)
```

Aquí la velocidad y la posición fueron definidas en su forma diferencial (para más detalle ver Feynman *et al*, 1999). El resultado de este pequeño script (*./proyectil.py*) se puede observar en la figura 5.

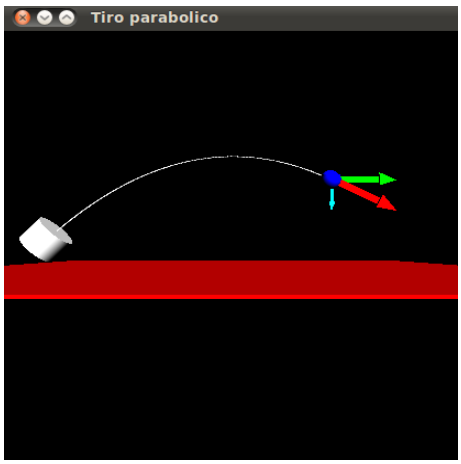

Figura 5. Resultado del script del Movimiento de un Proyectil. Se puede observar asociado a la bala del añon un vector rojo que representa la velocidad total, un vector verde que representa la velocidad en la componente x y un vector cyan que representa la velocidad en la componente y.

4.3 Energía vs Tiempo

Ahora vamos a hacer unas modificaciones al script para que al momento de correr la animación se genere una gráfica donde se pueda visualizar la energía cinética, potencial y total como función del tiempo. Para hacer esto vamos a extraer parte del primer código que escribimos. Pero antes de seguir recordemos un poco las ecuaciones de la energía.

Tenemos dos tipos de energía; una que esta asociada al movimiento de la partícula y otra asociada a su altura. La primera se denomina Energía Cinetica de un cuerpo y es aquella energía que posee debido a su movimiento. Se define como el trabajo necesario para acelerar un cuerpo de una masa determinada desde el reposo hasta la velocidad indicada. Una vez conseguida esta energía durante la aceleración, el cuerpo mantiene su energía cinética salvo que cambie su velocidad. Para que el cuerpo regrese a su estado de reposo se requiere un trabajo negativo de la misma magnitud que su energía cinética. Su expresión matemática es

$$K = \frac{1}{2}mv^2 \tag{5}$$

La segunda se llama Energía Potencial y es energía que mide la capacidad que tiene dicho sistema para realizar un trabajo en función exclusivamente de su posición o configuración. Puede pensarse como la energía almacenada en el sistema, o como una medida del trabajo que un sistema puede entregar. Su expresión matemática es

$$U = mgh \tag{6}$$

Por ultimo la Energía Total es la suma de la energía cinetica y la energía potencial

$$E = K + U \tag{7}$$

Los nuevos cambios se veran reflejados en 9 nuevas líneas. Las primeras líneas se colocaran abajo de la línea 6, y estan dirigidas a cambiar el color de fondo de la escena (comando background), y crear las condiciones para la gráfica:

```
scene.background = (1,1,1)

graph1 = gdisplay(x=0,y=0,width=600,height=450,

↪ title='E vs t', xtile='t', ytitle='E',foreground=color.black

↪ background=color.white)
```

Las siguientes líneas tendran como objetivo crear las tres líneas que representen la energía.

```
potencial = gcurve(gdisplay=graph1,color.blue)
cinetica = gcurve(gdisplay=graph1,color.red)
etotal = gcurve(gdisplay=graph1,color.green)
```

Abajo de la línea 14 definimos en valor de la masa.

```
m = 1
```

Abajo de la línea 36 (dentro del ciclo while) definimos las ordenes para graficar.

```
cinetica.plot(pos=(t,0.5*m*mag2(vel)))
potencial.plot(pos=(t,m*grav*proyectil.pos.y))
etotal.plot(pos=(t,m*grav*proyectil.pos.y + 0.5*m*mag2(vel)))
```

El resultado de este cambio (*./proyectil2.py*) se puede ver en la figura 6.

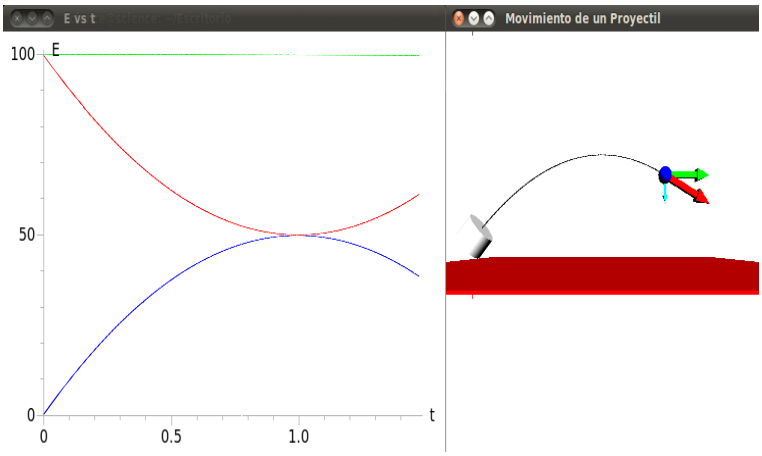

Figura 6. En la parte izquierda se muestra la grafica de E vs t. En la parte derecha de forma conjunta se muestra el movimiento de un proyectil.

5 Caida libre

Es hora de comenzar a soltarnos un poco.

Este script que vamos a construir tiene la intención de resolver el siguiente problema:

" Se sueltan dos bolas de masa m al mismo tiempo $t_o = 0$. Una bola se deja caer en forma vertical y la otra con un movimiento parabólico como se muestra en la figura 7. ¿Cuál llegara primero al suelo?" (para más detalle ver cap. 3, Movimiento en dos o tres dimensiones, Sears, 1999).

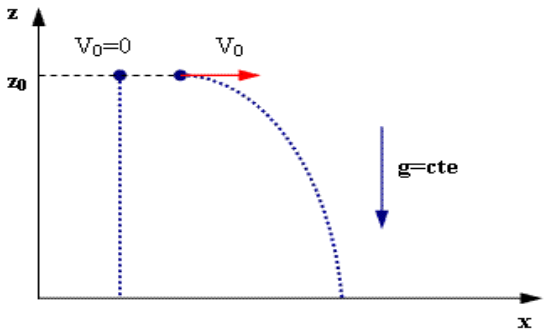

Figura 7. Diagrama que muestra la trayectoria de dos cuerpos en caida libre

Si ya quedo claro el problema comencemos a diseñar el código.

Como siempre las dos primeras líneas son

```
Redireccionamiento al interprete de python
Importación de las paqueterias de trabajo
```

Despues crearemos la escena donde trabajaremos

```
display(title='titulo', width=ancho,height=alto,background=(color de fondo))
```

El titulo *title* es una palabra o frase escrita entre comas altas, el ancho y la altura se dan en pixeles (por ejemplo 550), el código de color es RGB y se puede escribir con números enteros, decimales o fracciones que se dividan con respecto a 255 (ejemplo 220.0/255, 255.0/255, 130.0/255).

Ahora vamos a crear los elementos de nuestra escena (el piso y dos bolas de diferente color). Primero para el piso:

```
piso=box(pos(x,y,z),width,height,lenght,color,axis)
```

Los atributos de la caja son; *pos*, sirve para posicionar la altura en que se creara el plano; *width*, *height* y *length* son lo ancho, alto y profundo de la caja; *axis*, marca la posición (inclinación) de la caja. Para las bolas:

```
bola1=sphere(pos=(),radius=(),color)
```

Aquí aparece otro atributo nuevo; el atributo *radius* crea la dimensión de la bola (también puede operar en líneas). Ee este tipo de lenguajes es muy común utilizar funciones predefinidas. Este tipo de funciones se utilizan escribiendo el nombre de la función, un punto, y despúes su atributo. Como ejemplo, la función color ya esta definida, si deseamos invocar uno de los 8 colores básicos (negro, rojo, azul, blanco, amarillo, verde, cyan y magenta) escribimos *color.blue*.

Ahora, para darle un poco de originalidad al script, le vamos a pedir que aparte de hacer la simulación numérica, nos arroje cuatro valores de salida en todo momento (altura, rapidez, tiempo y alcance).

```
alcande = label(pos=(x,y,z),text=")
```

A partir de este momento estamos listos para comenzar a introducir las condiciones iniciales de nuestro sistema: Comencemos creando estos valores para la bola 1 (la que tiene velocidad inicial).

```
b1vx = 15
b1vy = 0
```

Y creando dos nuevos grupos de valiables, las cuales me serviran para determinar las posciones iniciales y finales:

```
b1x = bola1.pos.x
b1y = bola1.pos.y
pox1 = bola1.pos.x
poy1 = bola1.pos.y
```

Ya que en esta ocación tenemos definidos cuatro recuadros donde se tendrán los datos instantaneos de rapidez y tiempo, debemos definir un paso diferencial cuando el límite tiende a cero, esto es:

```
dt = 0.
```

Como siempre, la forma más sencilla de dotar de movimiento nuestra simulación es con un ciclo *while*

while 1:
rate(numero de cuadros)
dt +=0.005
if bola1.pos.y > 0.5:

Para programar las ecuaciones debemos de recordar un poco el comportamiento de este sistema dual de proyectiles en caida libre. Analizando fotos estroboscopicas de este sistema (figura 8), se sabe que en un instante dado, ambas bolas tienen la misma posición y, velocidad y, y aceleración y, a pesar de tener diferente posición x y velocidad x (Sears, 1999).

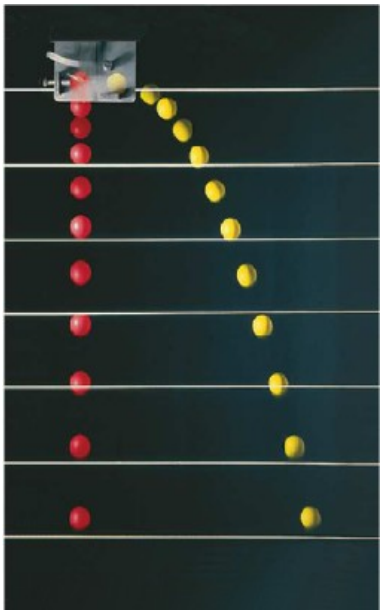

Figura 8. Posición de dos bolas en caida libre en diferentes tiempos.

Sabiendo lo anterior comencemos a nalizar el sistema por separado:

- Para la bola 1

Esta bola tiene una rapidez inicial en la componente x (igual a 15 m/s) y en$y = 0$. Analizando las ecuaciones (1-4) tenemos que

$$V_x = V_{ox} = 15$$

$$x = x_o + V_{ox}t$$
$$V_y = V_{oy} - gt = -gt$$
$$y = y_o + V_{oy} - \frac{1}{2}gt^2 = y_o - 4.9t^2$$

- Para la bola 2

$$V_x = 0$$
$$x = x_o = 0$$
$$V_y = V_{oy} - gt = -gt$$
$$y = y_o + V_{oy}t - \frac{1}{2}gt^2 = -4.9t^2$$

Programando estas ecuaciones tendremos que:

bola1.pos.y = $y_o - V_{oy}t - \frac{1}{2}gt^2$

bola1.pos.x = $x_o - V_{ox}t$

Como siempre, vamos a asociar una trayectoria a cada bola, y aprovechando vamos a generar una barra que marque la distancia de separación entre ambas bolas.

curve(pos=[(x_o, y_o, z_o),(x_f, y_f, z_f)],color=,radius=)

Ya teniendo esto vamos a enlazar los valores de salida a sus respectivos recuadros.

alcance.txt='Alcance' + str(round(x2-x1,2)) + 'm'

Para la rapidez conviene primero analizar que salida es la que queremos. En pasos anteriores hemos definido la componente de la velocidad en x y en y, sin embargo lo que nosotros queremos es la magnitud de la velocidad, en otras palabras lo que queremos es

$$rapide = \sqrt{V_{ox}^2 + V_{oy}^2}$$

Como $V_y = -gt$ nos queda

$$rapide = \sqrt{V_{ox}^2 + (9.8t)^2}$$

y para finalizar establecemos los parametros finales de cierre

```
b1x = bola1.pos.x
b1y = bola1.pos.y
```

El resultado de este programa al correr el codigo (*./proyectil3.py*) se muestra en la figura 9.

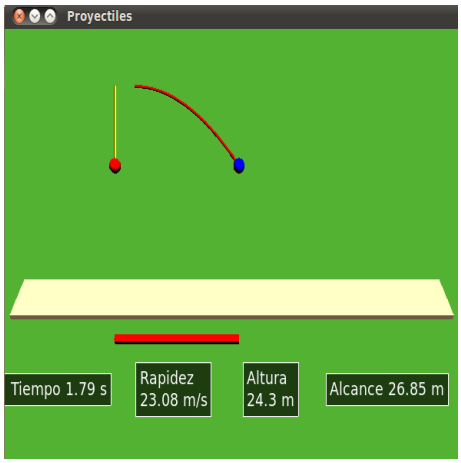

Figura 9. Resultado del script del sistema de dos bolas en caida libre

6 Vectores

Otro tema de importancia para la física es el de los vectores ya que con estos se pueden resolver los problemas del movimiento en dos y tres dimensiones. El primer ejemplo que trataremos aquí es de la suma de dos vectores utilizando el método del polígono.

El método del polígono para sumar vectores dice que, el vector resultante se obtiene dibujando cada vector a escala, colocando el origen de un vector en la punta de la flecha del otro hasta que todos los vectores queden representados. La resultante es la línea recta que se dibuja a partir del origen del primer vector hasta la punta del último (Tippens, 2001).

6.1 Vectores 2D

Suponga que una partícula sufre un desplazamiento $\vec{A}$ seguido de un desplazamiento $\vec{B}$, el resultado final es el mismo que si la partícula hubiera sufrido un solo desplazamiento $\vec{C}$ igual a la suma de la magnitud de $\vec{A} + \vec{B}$ (Sears *et al*, 1999).

Como ejemplo, podemos suponer que tenemos un avión que realiza los movimientos de la figura 10 (vistos desde la pantalla del controlador de vuelo)

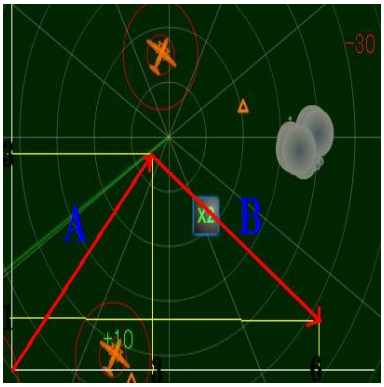

Figura 10. Las flechas rojas marcan el trayecto de un avión

Al analizar la pantalla con cuidado podemos concluir que

- El avión partio de las coordenadas $A_o(0,0)$ y esa parte del trayecto la terminó en $A_f(3,3)$
- El trayecto B lo inicio en $B_o(3,3)$ y lo terminó en $B_f(6,1)$
- El trayecto resultante es $\vec{C} = \vec{A} + \vec{B} = (3,3) + (6,1) = (9,4)$

Comencemos con el código de este problema utilizando el método del polígono:

```
Redirección al interprete de Python

Importar paqueterias

crear escena (con tamaño)

ponerle titulo a la escena

crear el rango de la escena

construir los valores de inicio y fin del vector 1

construir los valores de inicio y fin del vector 2

construir los valores de inicio y fin del vector resultante

resolver la magnitud del vector 1
```

resolver la magnitud del vector 2

resolver la magnitud del vector resultante | crear el eje x

crear el eje y

Dotar de movimiento el vector 1

Dotar de movimiento el vector 2

Dotar de movimiento al vector resultante

Escribir como salida en la Terminal, la magnitud del vector resultante

El resultado de este código se puede observar en la siguiente figura (./suma2vectores.py)

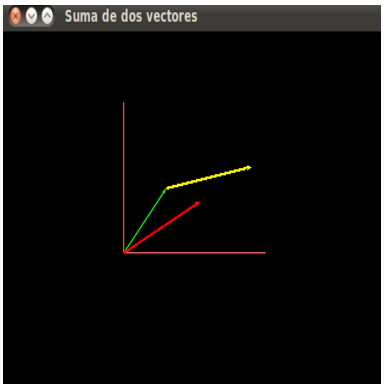

Figura 11. Suma de dos vectores donde se muestra la resultante en rojo

6.2 Vectores 3D

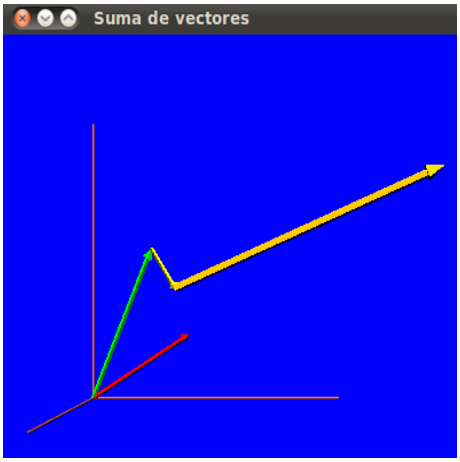

Figura 12. Suma de tres vectores en un plano tridimensional

La figura 12 muestra el resultado de un script que resuelve la suma de tres vectores por el método del polígono. Para hacer más divertido y útil este programa vamos a construirlo a partir de una necesidad.

Resuelve el siguiente problema de vectores y escribe su código (pero ahora que el resultado salga en la misma escena):

“ Un barco recorre 100 km hacia el norte durante el primer día de viaje, 60 km al nordeste el segundo día, y 120 km hacia el este el tercer día (figura 13). Encuentre el desplazamiento resultante con el método del polígono”.

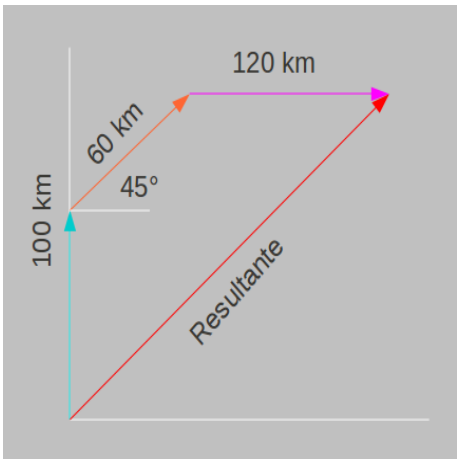

Figura 13. Trayectoria de recorrido del barco en 3 días.

7 Apéndice de códigos

7.1 Graficador

```
#! /usr/bin/env python

    #************************************
    from visual import *
    from visual.graph import *

    #introduciendo las funciones f(x)
    ecu1= raw_input("Escriba su ecuacion usando lenguaje python: ")
    ecu2= raw_input("Escriba su segunda ecuacion : ")

    #gcurve(x=x,y=0, radius=0.1, color=color.blue)
    f1=gcurve( color=color.cyan)
    f2=gcurve( color=color.yellow)

    for x in arange(0,20,0.1):
    f1.plot(pos=(x, eval(ecu1)))
    f2.plot(pos=(x, eval(ecu2)))
    #****************FIN**************************
```

7.2 Atomo de Helio

```
#! /usr/bin/env python
    from visual import *

    scene2 = display(title='ATOMO',x=0, y=0, width=600, height=500,center=(5,0,0),
background=(0.2,0.3,1))
    scene2.lights = [vector(6,0,2)]

    n1 = sphere(pos=(1,0,0), radius=1, color = color.green)
    n2= sphere(pos=(2,1,0), radius=1, color = color.green)

    p1 = sphere(pos=(2,0,0), radius=1, color = color.red)
    p2 = sphere(pos=(1,1,0), radius=1, color = color.red)

    e1 = sphere(pos=(15,7,0), radius=0.5, color = color.blue)
    e2 = sphere(pos=(-3,15,5), radius=0.5, color = color.blue)

    dt=0.1

    curva1= curve()
    curva2= curve()

    while 1:
    rate(20)
    dt=0.1

    e1.rotate(angle=-dt, origin=(0,0,0), axis=(-0.3,0.8,0.3))
    e2.rotate(angle=-dt, origin=(0,0,0), axis=(1,0.4,0.6))
```

```
curva1.append(pos=e1.pos)
curva2.append(pos=e2.pos)
#*************************FIN*****************************
```

7.3 Proyectil 1

```
#! /usr/bin/env python
from visual import *

scene = display()
scene.title = 'Movimiento de un Proyectil'
scene.autoscale = 0

pos = vector(-10,0,0)
vel = vector(10,10,0)
grav = 10
acel = vector(0,-grav,0)
dt = 1./300.
t = 0

proyectil = sphere(pos=pos,color=color.blue,radius=0.5)
suelo = box(pos = (0,-1,0),size = (25,0.1,25),color = color.red)
cannon = cylinder(pos = pos,axis = (1,1,0))
trayectoria = curve(color = color.white)

velocidadtotal = arrow(color = color.red,pos = proyectil.pos,axis = vel/3.)
velocidadx = arrow(color = color.green,pos = proyectil.pos,axis = (vel.x/3.,0,0))
velocidady = arrow(color = color.cyan,pos = proyectil.pos,axis = (0,vel.y/3.,0))

while pos.y >= 0:
vel = vel+acel*dt
pos = pos+vel*dt
trayectoria.append(pos)
proyectil.pos = pos
velocidadtotal.pos = pos
velocidadx.pos = pos
velocidady.pos = pos
velocidadtotal.axis = vel/3.
velocidadx.axis = vector(vel.x/3.,0,0)
velocidady.axis = vector(0,vel.y/3.,0)
t = t+dt
rate(50)
#*****************FIN*************************
```

7.4 Proyectil 2

```
#! /usr/bin/env python

    from visual import *
    from visual.graph import *

    scene = display()
    scene.title = 'Movimiento de un Proyectil'
    scene.autoscale = 0

    # linea 1 nueva
    scene.background = (1,1,1)

    # linea 2 nueva
    graph1 = gdisplay(x=0, y=0, width=600, height = 450, title = 'E vs t',
xtitle = 't', ytitle = 'E', foreground = color.black, background = color.white)

    # Linea 3 nueva
    potencial = gcurve(gdisplay = graph1, color = color.blue)
    cinetica = gcurve(gdisplay = graph1, color = color.red)
    etotal = gcurve(gdisplay = graph1, color = color.green)

    pos = vector(-10,0,0)
    vel = vector(10,10,0)
    grav = 10

    # Linea 4 nueva
    m = 1

    acel = vector(0,-grav,0)
    dt = 1./300.
    t = 0
    proyectil = sphere(pos=pos,color=color.blue,radius=0.5)
    suelo = box(pos = (0,-1,0),size = (25,0.1,25),color = color.red)
    cannon = cylinder(pos = pos,axis = (1,1,0))
    trayectoria = curve(color = color.black)
    velocidadtotal = arrow(color = color.red,pos = proyectil.pos,axis = vel/3.)
    velocidadx = arrow(color = color.green,pos = proyectil.pos,axis = (vel.x/3.,0,0))
    velocidady = arrow(color = color.cyan,pos = proyectil.pos,axis = (0,vel.y/3.,0))

    while pos.y >= 0:

    vel = vel+acel*dt
    pos = pos+vel*dt
    trayectoria.append(pos)
    proyectil.pos = pos
    velocidadtotal.pos = pos
    velocidadx.pos = pos
    velocidady.pos = pos
    velocidadtotal.axis = vel/3.
```

```
velocidadx.axis = vector(vel.x/3.,0,0)
velocidady.axis = vector(0,vel.y/3.,0)

# Linea 5 nueva
cinetica.plot(pos=(t,0.5*m*mag2(vel)))
potencial.plot(pos=(t,m*grav*proyectil.pos.y))
etotal.plot(pos=(t,m*grav*proyectil.pos.y+0.5*m*mag2(vel)))

t = t+dt
rate(50)
#*****************FIN*************************
```

7.5 Proyectil 3

```
#! /usr/bin/env python
from visual import *

# Crear la escena
display(title="Proyectiles", width = 600, height=550,background=(0.33,0.7,0.2))

# Crear los tres elementos principales
piso = box(pos=(0,-2,0), width =110, height=1, length=10, color= (220.0/255,
181.0/255,134.0/255), axis= (0,0.2,-0.2))
bola1 = sphere(pos=(-25,40,0), radius=1.5, color = color.blue)
bola2 = sphere(pos=(-30,40,0), radius=1.5, color = color.red)

# Crear espacio para salida de datos
alcance=label(pos=(40,-20,0), text="Alcance X")
altura= label(pos=(10,-20,0), text="Altura")
rapidez= label(pos=(-15,-20,0), text="Rapidez")
tiempo= label(pos=(-45,-20,0), text="Tiempo")

#definicion de la velocidad inicial: proyectil lanzado horizontalmente
bola1_velocity_y=0
bola1_velocity_x= 15
b1x=bola1.pos.x
b1y=bola1.pos.y
pox1= bola1.pos.x
poy1= bola1.pos.y

#definicion de la velocidad inicial: proyectil lanzado horizontalmente
bola2_velocity_y=0
bola2_velocity_x= 0
b2x=bola2.pos.x
b2y=bola2.pos.y
pox2= bola2.pos.x
poy2= bola2.pos.y

# Definiendo el paso diferencial
dt = 0.
```

```
    # Dotando de movimiento esta animacion
    while 1:
    rate (20)
    dt += 0.005
    if bola1.pos.y >0.5:

    #Programando ecuaciones de proyectiles
    bola1.pos.y= poy1-bola1_velocity_y*dt-4.9*dt**2
    bola1.pos.x = pox1 +bola1_velocity_x*dt
    bola2.pos.y= poy2-bola2_velocity_y*dt-4.9*dt**2
    bola2.pos.x = pox2 +bola2_velocity_x*dt

    # Definiendo la caracteristicas de las trayectorias
    #Trayectoria: bola 1
    curve(pos=[(b1x,b1y,0),(bola1.pos.x, bola1.pos.y,0)], color = color.red, ra-
dius=0.3)

    #Alcance en X
    curve(pos=[(-30,-10,0),(bola1.pos.x, -10,0)], color = color.red, radius=1)

    #Trayectoria: bola 2
    curve(pos=[(-30,b1y,0),(-30, bola1.pos.y,0)], color = (255.0/255,255.0/255,1.0/255),
radius=0.3)

    # Dando los valores de salida
    alcance.text="Alcance " + str(round(bola1.pos.x-pox1,2)) + " m"
    rapide = (bola1_velocity_x**2 + (9.8*dt)**2)**0.5
    altura.text="Altura \n" + str(round(bola1.pos.y ,2)) + " m"
    rapidez.text="Rapidez \n" + str(round(rapide ,2)) + " m/s"
    tiempo.text = " Tiempo " + str(round(dt,2)) + " s"

    b1x=bola1.pos.x
    b1y=bola1.pos.y
    #**********FIN**********
```

7.6 Suma de dos vectores

```
#! /usr/bin/env python
    from visual import *
    #************************************

    scene = display(width=500, height=400,center=(5,0,15))
    scene.title = 'Suma de dos vectores'
    scene.fullscreen = 0
    scene.range = (5,1,1)
    scene.autoscale=0

    vector1=arrow(pos=(0,0), axis=(3,3), shaftwidth=0.1, length= 0.01, color=color.green)
    vector2=arrow(pos=(3,3), axis=(6,1), shaftwidth=0.1, length= 0.01, color=(1,1,0))
```

```
    vector_resultante=arrow(pos=(0,0,0), axis=(9,4), shaftwidth=0.01, length=
0.01, color=color.red)

    axis1=mag(vector(3,3))
    axis2=mag(vector(6,1))
    axis3=mag(vector(9,4))

    eje_x= curve(pos=[(0,0,0), (10,0,0)], radius=0.05, color=(0.7,0.3,0.3))
    eje_y= curve(pos=[(0,0,0), (0,7,0)], radius=0.05, color=(0.7,0.3,0.3))

    i=0
    while i <20:
    rate(10)
    vector1.length += axis1/20.0
    i+=1

    i=0
    while i <20:
    rate(10)
    vector2.length += axis2/20.0
    i+=1

    i=0
    while i <20:
    rate(10)
    vector_resultante.length += axis3/20.0
    i+=1

    print 'la magnitud del vector resultante es = ', axis3
    #**********************FIN**************
```

7.7 Suma de tres vectores

```
#! /usr/bin/env python
    from visual import *

    scene2 = display(title='Suma de tres vectores',x=0, y=0, width=600, height=600,center=(10,0,3),
background=(0,0,1))

    # Definiendo los vectores a sumar y su resultante
    vector1=arrow(pos=(0,0,0), axis=(3,5,2), shaftwidth=0.1, length= 0.01, co-
lor=color.green)
    vector2=arrow(pos=(3,5,2), axis=(2,-2,4), shaftwidth=0.1, length= 0.01, co-
lor=(1,1,0))
    vector3=arrow(pos=(5,3,6), axis=(8,3,1), shaftwidth=0.1, length= 0.01, co-
lor=color.orange)
    vector_resultante=arrow(pos=(0,0,0), axis=(13,6,7), shaftwidth=0.01, length=
0.01, color=color.red)

    axis1=mag(vector(3,5,2))
```

```
axis2=mag(vector(2,-2,4))
axis3=mag(vector(8,3,1))
axis4=mag(vector(13,6,7))

#graficando sistema de coordenadas
eje_x= curve(pos=[(0,0,0), (10,0,0)], radius=0.05, color=(0.7,0.3,0.3))
eje_y= curve(pos=[(0,0,0), (0,10,0)], radius=0.05, color=(0.7,0.3,0.3))
eje_z= curve(pos=[(0,0,0), (0,-1,5)], radius=0.05, color=(0.7,0.3,0.3))

# Valores de salida
i=0
while i <20:
rate(10)
vector1.length += axis1/20.0
i+=1

i=0
while i <20:
rate(10)
vector2.length += axis2/20.0
i+=1

i=0
while i <20:
rate(10)
vector3.length += axis3/20.0
i+=1

i=0
while i <20:
rate(10)
vector_resultante.length += axis4/20.0
i+=1

resultante = label(pos=(10,-5,10), text = "Resultante")
resultante.text = 'La magnitud de la resultante es \n' + str(round(axis4,2))
+ ' km'
#******************FIN*******************
```

Referencias

[1] Chabay, R. W. and Sherwood, B. A. (2007) Research-based Reform of University Physics, chapter Matter and Interactions. PER-Central.

[2] Feynman, R.P., Leighton, R.B. & Sands, M. (1998) Física. Vol I. Pearson Educación.

[3] García, C.P. (2008) Proyecto Physthones. Simulaciones Físicas en Visual Python (0.1). Un libro libre de Alqua.

[4] García, J.H. & Ferreiro, A. (2005) Representación gráfica 2D: Matplotlib. www.linux-Magazine.es

[5] Marzal, A., Gracia, I. (2003) Introducción a la Programación con Python. Departamento de Lenguajes y Sistemas Informaticos. Universitat Jaume I.

[6] Mireles, L.O. (2011) Brevisimo manual para el tratamiento inicial de datos utilizando la plataforma Linux.

[7] Piskunov, N. (1978) Calculo Diferencial e Integral. Ed. MIR. Moscu.

[8] Sanders, D.P. (2010) Notas del curso: Programación en Python. Curso de actualizaión docente. Facultad de Ciencias.

[9] Sears, F.W., Zemansky, M.W., Young, H.D. & Freedman, R.A. (1999) Física Universitaria. Pearson Educación.

[10] Tippens, P.E. (2001) Física: Conceptos y aplicaciones. Ed. McGraw Hill