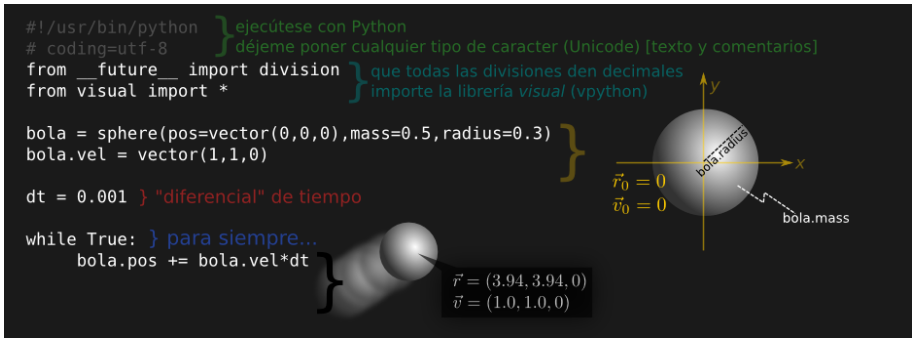
Simulaciones

9 de septiembre de 2011

Parte del "Taller de Programación Científica: Simulaciones"
IFT/CONFER – PCF-LIAP, Universidad de Costa Rica.
Expositor: Guillermo Andrée Oliva Mercado

1 El "hola mundo" de las simulaciones

- Con los principios de programación listos, podemos empezar a programar simulaciones. Hay que empezar con el "hola mundo" de las simulaciones: hacer que una bola se mueva con velocidad constante.



- Al ejecutar el código, no se ve nada. Hay que agregar un

```
rate(100)
```

  después del `while True`. ¡Lo que pasa es que la simulación está ejecutándose a la velocidad del procesador! Eso es muy muy rápido. Hay que pedirle al procesador que lo haga más lento para que podamos verlo.

- Veremos que la bola se encoje en lugar de avanzar. Eso es porque por defecto, *visual* ajusta el zum[1] de la escena para que quepan todos los objetos. Fijamos un tamaño de la escena con

```
scene.range = (2,2,2)
```

[1] Sí, la versión españolizada de *zoom* es zum.

2 Un proyectil

El código está en el archivo `proyectil.py` .

- Tal vez usted se esté preguntando que dónde está el $-1/2gt^2$ y etc. que aparecen cuando se resuelve analíticamente este problema. Es decir, ¡¿dónde están las integrales?! Lo que pasa es que hemos resuelto la ecuación diferencial del movimiento de forma numérica. La integración ocurre cuando usted hace la suma del valor que tenía antes más la derivada por un "diferencial". Y como la ecuación es de segundo orden, hemos integrado dos veces para obtener la posición.
- Introducimos el objeto tipo *box*, que nos representa una caja, que la hemos usado para el piso en este caso.

```
caja = box(width=W,height=H,length=L,pos=vector(a,b,c))
```

- ■ *Ejercicio.* ¿Qué pasa al ejecutar `caja.axis = vector(1,1,0)` ?
- Lo que hicimos para detener la bola al chocar con el piso es una colisión completamente inelástica.

3 Fuerza de Lorentz

El código está en el archivo `lorentz.py` .

- Hemos usado la función *cross*, que calcula el producto cruz de vectores tridimensionales. Busque más formas de trabajar con vectores en la documentación de Vpython. [1]
- En este caso, a diferencia de los dos ejemplos anteriores, era necesario poner la masa del objeto, y calcular la aceleración como

```
bola.acc = bola.force / bola.mass
```

- ■ Experimente con condiciones iniciales distintas para esta simulación. ¿Qué pasa si ponemos velocidad inicial y quitamos el campo eléctrico?

4 Esquema de una simulación

Hasta el momento hemos hecho dos simulaciones sencillas. Ahora resumamos lo que aprendimos sobre simulaciones.

- *Hay que* modelar *el problema con objetos.* Es lo que hacen los físicos todo el tiempo... modelar. Dijimos que una bola era una esfera, o que el campo magnético era un vector... Las cuerdas, resortes, cajas, etc., etc., también deben ser modeladas con objetos. En la sesión de la sintaxis de Python, modelamos una partícula (sin forma siquiera).

- Lo anterior es generalmente subjetivo; hay muchas maneras de modelar un problema, y hasta adónde usted debe modelarlo. Es análogo al problema de *¿qué es una partícula?*
- Una buena simulación debería ser aquella en la que la parte física conceptual de un problema esté metida en el código. Y en lo posible, las matemáticas deberían "resolverse solas". Es decir, ud no debería resolver una ecuación diferencial analíticamente (a mano) para luego sustituir valores en la simulación. Debería procurar que el programa resuelva las ecuaciones solo.
- Los lenguajes como C o Bash, que no son orientados a objetos, no están diseñados para que ud modele así la física conceptual de un problema. Note sin embargo, lo mucho que se parece un problema resuelto con C a cómo usted resuelve analíticamente sus problemas de física.

  ```
  KE_particle = 0.5*mass*speed**2 # Python sin objetos (o C)
  ```

  $$KE_{caja} = \frac{1}{2}mv^2$$

- Tema de conversación: creación de nuevas notaciones para la física que nos permitan "meter" la parte conceptual en las ecuaciones. Algo como...

  ```
  particle.KE = 0.5*particle.mass*mag(particle.vel)**2
  ```

  pero analítico. ¿■$_{KE} = \frac{1}{2}$■$_m||$■$_{\vec{v}}||^2$? Especialmente útil cuando dos cuerpos interactúan.

- *Condiciones iniciales.* Al crear el objeto, hay que establecer su masa, velocidad, posición, etc., que pueden variar con el tiempo más adelante.
- *El "diferencial" de tiempo.* Realmente es un Δt o un δt. Elija el valor que más le convenga, para que la simulación vaya a la escala temporal que necesita.
- *El "while True".* Esta es la parte que realmente hace la animación: el cambio constante de los valores que se dieron inicialmente.
- *La solución de las ecuaciones diferenciales.* Generalmente, usted se planteará ecuaciones diferenciales que hay que resolver numéricamente en cada paso. Puede haber otras ecuaciones o sistemas de ecuaciones que también debe resolverse constantemente.
- *La demás variables.* Una vez resueltas las ecuaciones del movimiento, pueden calcularse otras variables como energía cinética o potencial.

- *Las condiciones.* A veces, usted quiere que la simulación cambie de rumbo o se detenga cuando pase algo. Por ejemplo, cuando el proyectil tocó el piso.

- *Sistemas de coordenadas.* En algunos problemas puede ser que usted necesite otro sistema de coordenadas diferente al cartesiano que se mostrará en la pantalla. En este caso, debe transformar las coordenadas para obtener la posición en cada paso de la simulación. Siempre que pueda, resuelva *vectorialmente* las ecuaciones del movimiento. Es muchísimo más fácil que pensar en sistemas de coordenadas. Por ejemplo, una simulación de un planeta alrededor de su estrella *no requiere* de otro sistema de coordenadas diferente al cartesiano para ser resuelto. (¡Pida más talleres!)

- *Las partes meramente estéticas.* A veces necesitará etiquetas, imágenes, flechas, etc., que no forman parte de la simulación. Como sugerencia, sepárelas de las partes importantes del código y comente que solo son estéticas.

5 Ejercicio: oscilador armónico simple

¿Cuál es por mucho, la ecuación que más aparece en casi todas las teorías centrales de la física? Por supuesto, el oscilador armónico simple. En mecánica (de partículas, de sólidos), en circuitos, electromagnetismo, ondas, mecánica cuántica, etc... Y usted va a programar una simulación del oscilador armónico a continuación.

Ejercicio secreto. Después se subirá a la página del taller la solución.

References

[1] Documentación de Vpython. Puede leerse en internet desde http://vpython.org/contents/docs/visual/index.html

Licencia