Revista Ingeniería, Investigación y Desarrollo, Vol. 16 Nº 1, pp. (72-82), 2016

DESARROLLO DE APLICACIONES EN PYTHON PARA EL APRENDIZAJE DE FÍSICA COMPUTACIONAL

(Development of Python applications for learning computational physics)

[1]Jesús Daniel Arias Hernández, [1]Andrés Fernando Jiménez López, [1]Hernán Oswaldo Porras Castro
[1]Departamento de Matemáticas y Física, Unillanos, Grupo inv. Sistemas Dinámicos, jesus.arias@unillanos.edu.co, ajimenez@unillanos.edu.co, hernan.porras@unillanos.edu.co
Universidad de los Llanos

(Recibido el 16 de agosto de 2015 y aceptado el 20 de octubre de 2015)

Resumen
Este artículo describe una aplicación desarrollada para el aprendizaje de algoritmos de simulación basados en conceptos de mecánica clásica. Los estudiantes de Ingeniería Electrónica y de Ciencias de la Computación de la Universidad de los Llanos estudian la física computacional usando cinemática de partículas (CP), como una de las actividades del grupo de investigación Sistemas Dinámicos. Python, el lenguaje de programación seleccionado, facilita portabilidad y el acceso a las librerías necesarias para la representación de partículas. Las principales librerías de Python usadas en este ejercicio son: matplotlib, numpy, PyQt4, scipy, Tkinter y VPython. Estas librerías permiten la simulación de movimiento uniforme, movimiento lineal acelerado, caída libre y movimiento de proyectiles. Además, son útiles para la generación de interfaces gráficas de usuario para mostrar los datos en tablas y gráficos. Las GUIs fueron implementadas usando las librerías Tkinter y PyQt4, donde esta última facilita el desarrollo con la ayuda de herramientas del *software* Qt Designer.

Palabras clave: física computacional, movimiento, Python, simulación, software libre.

Abstract
This paper describes an application developed for learning simulation algorithms based on classical mechanics concepts. Electronics Engineering and Computer Science students at the Universidad de los Llanos study computational physics, using Particle kinematics (PK), as one activity of Dynamic Systems research group. Python, the programming language selected, facilitated the portability and access to the libraries necessary for representing particles. The main Python Libraries used in this process are matplotlib, numpy, PyQt4, scipy, Tkinter and VPython. These libraries allow the simulation of uniform motion, linear accelerated movement, objects in free fall and projectile motion. Additionally they are useful for generating GUIs to show data in tables and graphics. The GUIs were implemented using the Tkinter and PyQt4 libraries, where the latter facilitated the development with the help of Qt Designer software Tools.

Keywords: computational physics, free software, motion, Python, simulation.

1. INTRODUCCIÓN

Los investigadores de la enseñanza expresan que los métodos de aprendizaje constructivista son los más eficientes en este campo, ya que conducen a la mayor tasa de retención. Los constructivistas piensan que el aprendizaje se construye del propio conocimiento, es decir, los estudiantes son participantes activos en el proceso de aprendizaje buscando encontrar el significado en sus propias experiencias (Sener, 1997). El aprendizaje activo, el aprendizaje por descubrimiento y la construcción del conocimiento son variaciones de constructivismo. En todas estas, un estudiante tiene la libertad de exploración dentro de un marco o estructura establecida. Los profesores constructivistas no se comportan como personas que saben todo sobre el

tema, sino que actúan como facilitadores que estimulan a los estudiantes para que descubran principios por ellos mismos y construyan el conocimiento a través de sus intentos de resolver problemas reales (Panou, 2008).

Los estudiantes deben percibir e interpretar constantemente el proceso de instrucción, para aumentar el grado de asimilación de conocimiento. Por tal motivo, los instructores son responsables de proporcionar a los alumnos ambientes amigables de aprendizaje que ofrezcan opciones, en el sentido de lograr la interacción social, el descubrimiento de grupo, la investigación y la reflexión individual. La investigación sobre el aprendizaje muestra que es necesario que los estudiantes construyan su propia comprensión de las ideas científicas dentro del marco de sus conocimientos existentes (Bransford, Brown & Cocking, 2004). A fin de permitir a los estudiantes construir su propia comprensión de ideas científicas, ellos deben ser alentados a participar activamente en el tema, para que puedan tener la oportunidad de aprender a través de ese compromiso. Los estudiantes se benefician de preguntas de predicción en la identificación de lo que es importante y la construcción de una estructura mental para la evaluación de los fenómenos. Este marco ayuda a los estudiantes a eliminar detalles innecesarios y especificar los componentes fundamentales que se han de memorizar (Zamarro, Molina & Núñez, 2004).

En este punto, es evidente que la importancia de la visualización durante la actividad de enseñanza no se debe ignorar y las clases deben incluir algunos recursos visuales, como simulaciones de algunos fenómenos físicos y experimentos en un laboratorio. Es más fácil recordar una película o un juego de vídeo, que todo lo que se pueda escribir en un tablero de clases. Los estudiantes de hoy están muy familiarizados con el uso de los computadores, y han crecido con el Internet y la cultura de los videojuegos, así que es una excelente idea involucrar a los estudiantes en la física mediante el uso de modelos visuales por computador, que simulen la apariencia y el movimiento, para brindar orientación e incentivar al mismo tiempo la innovación y la formación.

Actualmente, el mundo de la física se enfrenta a la incomprensión de las personas, debido a que para muchos es difícil entender la belleza de esta ciencia; pero la informática ha llegado para dar una luz de entendimiento, con programas y herramientas visuales que han comenzado a apoyar fuertemente en su enseñanza. El incremento en el poder computacional ha permitido que la academia desarrolle la educación en física computacional buscando la integración de la informática con la física, para obtener mejores resultados en el aprendizaje que la forma habitual de dictar los cursos. La educación en física computacional presenta un camino para seguir, pues integra las experiencias para estimular y activar en los estudiantes de ingeniería el deseo de generar una base sólida del conocimiento, útil para su experiencia profesional (Yasar & Landau, 2003).

Las ciencias computacionales son el origen de la física computacional, mediante la combinación de la física, la matemática aplicada y la informática, para solucionar problemas científicos. Se busca con estas aplicaciones fomentar el contacto alumno-profesor, la reciprocidad y la cooperación de los estudiantes, el aprendizaje activo, dar retroalimentación inmediata, destacar la importancia de pasar tiempo en una tarea, comunicar las expectativas y respetar la diversidad de talentos y formas de aprender (Chickering & Ehmann, 1996). El estudiante trabaja y discute sus proyectos con el profesor, para luego escribir un informe con secciones para ecuaciones, problemas, algoritmos, lista de códigos, visualización, discusión y crítica del experimento simulado. La visualización de los resultados es importante para todas las clases, y esto se hace usando paquetes como Maple, Mathematica, PtPlot, gnuplot, AceGr, u OpenDX para 2D, 3D, y gráficos animados (Landau, Páez & Bordeianu, 2010).

La física computacional es una gran herramienta para el estudio, enseñanza e investigación de fenómenos físicos (Egas, 2014). Pero también se sabe de experiencias en otras ramas de las ciencias, que utilizan los conocimientos computacionales, como por ejemplo: en biología (López, Narváez & Garzón, 2012), química (Vílchez, Marzocchi, Beldoménico & Vanzetti, 2013), matemáticas (Bustacara, 2010), psicología (García, Martín & Gutiérrez, 2010), ingeniería (Vargas & Murcia, 2005) y otras (Overholt, 2010).

Los estudiantes necesitan considerablemente más herramientas computacionales que sus predecesores, si quieren diferenciarse de sus competidores en cualquier profesión. Este artículo resume la estrategia para la combinación de ambientes de *software* y tópicos de física computacional (Yasar, 2006). El método asegura que los estudiantes adquieran un nivel mínimo de competencia en programación, con el objeto de construir sobre esta

una fundamentación que los prepare para enfrentarse al trabajo. Se busca desarrollar una herramienta de bajo costo que sirva para estudiantes y profesores, utilizando el lenguaje de programación Python, que ha incursionado con gran fuerza en las ciencias, por su versatilidad y la amplia gama de librerías especializadas que ofrece.

Se utiliza la librería Visual Python, para trabajar simulaciones de forma más vistosas a través del manejo de espacios en entornos 3D. Además de esta librería, se usó PyQt4 para el desarrollo de la interfaz gráfica de usuario. El ejercicio de aprendizaje se hizo en el grupo de estudio de física computacional, dentro del proyecto titulado: *Simulaciones de modelos de mecánica clásica mediante la implementación del software libre,* desarrollado por el grupo de investigación Sistemas Dinámicos de la Universidad de los Llanos.

2. MÉTODOS Y PROCEDIMIENTOS

Se utiliza Python como lenguaje de programación, por ser interpretado, multiplataforma, multiparadigma, de alto nivel y de fuente abierta. Con esta herramienta se implementaron tres procedimientos para el aprendizaje de algoritmos de simulación en Python; el primero, mediante un script de Python; el segundo ejercicio consistió en el desarrollo de una interfaz gráfica de usuario para establecer parámetros de la simulación, usando las librerías del ejercicio anterior, además de *PyQt4* para la interfaz gráfica; como tercer y último ejercicio se hizo una interfaz gráfica de usuario en Tkinter, que permite la conexión con VPython para realizar simulaciones gráficas en 3D.

2.1 Simulación mediante Script de Python

El programa utiliza las librerías: *numpy,* para el manejo de vectores y matrices; *matplotlib,* para gráficos en dos dimensiones; *math,* para operaciones matemáticas; *sys* y *os*, para uso del sistema operativo.

Los experimentos para simular en el programa son: el movimiento rectilíneo uniforme, movimiento uniformemente acelerado, caída libre y el movimiento parabólico. De los cuales se definieron los parámetros físicos y matemáticos útiles para la generación de los algoritmos. Para establecer los parámetros básicos de las simulaciones se tuvo en cuenta el comportamiento ideal correspondiente a cada movimiento por estudiar. Para realizar las simulaciones computacionales se tomaron las ecuaciones de la cinemática de una forma generalizada tanto para el $eje\ x$, ecuación (1) como para el $eje\ y$, ecuación (2).

$$x_f = x_i + v_{x_i}\, t \pm \frac{1}{2} a_x\, t^2 \qquad (1)$$

$$y_f = y_i + v_{y_i}\, t \pm \frac{1}{2} a_y\, t^2 \qquad (2)$$

Donde x_f y y_f son las posiciones finales, v_{x_i} y v_{y_i} son las velocidades iniciales, a_x y a_y son las aceleraciones y t el tiempo, definidos tanto para el $eje\ x$, como para el $eje\ y$.

Con esta forma generalizada se puede simular cualquier tipo de movimiento, ya sea rectilíneo o curvo, solamente declarando nulos los parámetros que no se necesiten. En este sentido, para la simulación de eventos físicos mediante física computacional se necesita definir condiciones iniciales, vectores de posición, velocidad y aceleración para cada eje coordenado. En donde se puede establecer en cada instante un vector de coordenadas (x, y) para cada variable en el espacio y el tiempo.

$$\vec{r} = \vec{r}\,(x, y, t) \qquad (3)$$

$$\vec{v} = \vec{v}\,(x, y, t) \qquad (4)$$

$$\vec{a} = \vec{a}\,(x, y, t) \qquad (5)$$

Los parámetros, si se definen para una única dimensión, es decir, en donde solo se necesita una coordenada espacial para determinar la posición de una partícula, pueden representarse como:

$$v(t) = \frac{dx(t)}{dt} \qquad (6)$$

$$a(t) = \frac{dv(t)}{dt} \qquad (7)$$

Siendo la posición instantánea $x(t)$, la velocidad $v(t)$ y la aceleración $a(t)$ de una partícula usando el lengua-

je del cálculo diferencial. Estas cantidades se conocen como las cantidades cinemáticas, porque describen el movimiento sin tener en cuenta sus causas. Ahora, se conoce que la fuerza neta que actúa sobre una partícula determina su aceleración. La segunda ley de Newton del movimiento dice que:

$$a(t) = \frac{F(x, v, t)}{m} \qquad (8)$$

Donde F es la fuerza neta y m es la masa inercial. En general, la fuerza depende de la posición, la velocidad y el tiempo. Por lo cual, la descripción del movimiento de una partícula requiere la solución de dos ecuaciones acopladas de primer orden, que por consiguiente corresponden a:

$$\frac{dv(t)}{dt} = \frac{d^2x(t)}{dt^2} = \frac{F}{m} \qquad (9)$$

Para estudiar la cinemática, los métodos aplicados en esta actividad de grupo de estudio son: Euler, Euler Richardson y Runge Kutta. Para el caso del método de Euler, si Δt es el intervalo de tiempo entre pasos sucesivos y a_n , v_n y x_n son los valores de aceleración, velocidad y posición para el tiempo $t_n = t_0 + n\Delta t$, por ejemplo $a_n = a_n(x_n, v_n, t_n)$, se puede hacer la generalización del método de Euler de forma tal que:

$$v_{n+1} = v_n + a_n \Delta t \qquad (10)$$

y

$$x_{n+1} = x_n + v_n \Delta t \qquad (11)$$

Donde v_{n+1}, es la velocidad al final del intervalo que depende de a_n; de la misma manera x_{n+1}, es la posición al final de un intervalo que depende de v_n. El algoritmo para obtener una solución numérica de la ecuación diferencial no es única, y existen muchos algoritmos que reducen a la misma ecuación diferencial en el límite $\Delta t \to 0$. La función en Python, útil para el procedimiento de Euler, es:

```
1.  def Euler(x, t, dt, func):
2.      x_sig =x+func(x,t)*dt
3.      return x_sig
```

Donde $func$ es una función que proviene de la ecuación característica del movimiento a estudiar.

El método de Euler Richardson sugiere que es mejor evaluar la velocidad en la mitad del intervalo anterior que al inicio o al final del mismo, debido a que el algoritmo se utiliza para fuerzas dependientes de la velocidad. El algoritmo consiste en usar el método para encontrar la posición intermedia x_{mid} y la velocidad v_{mid} para el tiempo $t_{mid} = t + \Delta t/2$. Luego, hallar la Fuerza, $F(y_{mid}, v_{mid}, t_{mid})$ y la aceleración a_{mid} para t_{mid}. Lo cual se puede representar como:

$$a_n = \frac{F(x_n, v_n, t_n)}{m} \qquad (12)$$

$$v_{mid} = v_n + \frac{1}{2} a_n \Delta t \qquad (13)$$

$$v_{mid} = v_n + \frac{1}{2} a_n \Delta t \qquad (13)$$

$$x_{mid} = x_n + \frac{1}{2} v_n \Delta t \qquad (14)$$

$$a_{mid} = \frac{F\left(x_m, v_m, t + \frac{1}{2}\Delta t\right)}{m} \qquad (15)$$

$$v_{n+1} = v_n + a_{mid} \Delta t \qquad (16)$$

Y

$$x_{n+1} = x_n + v_{mid} \Delta t \qquad (17)$$

Para el caso de Euler_Richardson, la función utilizada en Python se muestra a continuación:

```
4.  def euler_richar(x,t,dt,func):
5.    mitaddt = 0.5 * dt
6.    xmedio = x + mitaddt * func(x,t)
7.    x_sig = x + func(xmedio, t + mitaddt)*
    dt
8.      return x_sig
```

Runge Kutta es un procedimiento para la resolución de ecuaciones diferenciales, en donde una ecuación diferencial del tipo $x' = f(t, x)$, con condición inicial $x(t_0) = x_0$ se expresa como en la ecuación (18), según el método de Runge Kutta para este problema.

$$x_{j+1} = x + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad (18)$$

Donde

$$\begin{cases} k_1 = f(t_j, x_j) * dt \\ k_2 = f\left(t_j + \frac{1}{2}dt, x_j + \frac{1}{2}k_1\right) * dt \\ k_3 = f\left(t_i + \frac{1}{2}dt, x_i + \frac{1}{2}k_2\right) * dt \\ k_4 = f(t_i + dt, x_i + k_3) * dt \end{cases}$$

De forma tal, que el valor (x_{j+1}) se determina por el valor existente en (x_j) más una pendiente, que es un promedio ponderado de pendientes al comienzo del intervalo (k_1), en la mitad del intervalo (k_2, k_3) y al final del mismo (k_4), siendo las pendientes intermedias las que tienen mayor peso. La siguiente es una función en Python que permite encontrar el siguiente valor del movimiento mediante el procedimiento de Runge Kutta.

```
9.  def runge_kutta(x,t,dt,func):
10.     k1 =func(x,t)*dt
11.     k2 =func(x+0.5*k1,t+0.5*dt)*dt
12.     k3 =func(x+0.5*k2,t+0.5*dt)*dt
13.     k4 =func(x+k3,t+dt)*dt
14.     x_sig = x + 1/6.0*(k1+2*k2+2*k3+k4)
15.     return x_sig
```

Una vez definidas las funciones para los tres procedimientos, se realiza un código para simular cada uno de los movimientos propuestos. El código del programa generado mediante el script de Python se muestra a continuación:

```
16. N=30
17. tau=3.0
18. dt=tau/float(N-1)
19. x0=0.0
20. v0=0.0
21. gravedad=9.8
22. time=numpy.linspace(0,tau,N)
23. x=numpy.zeros([N,2])
24. x[0,0]=x0
25. x[0,1]=v0
26. def Ecu(state, time):
27.     g0=state[1]
28.     g1=gravedad
29.     return numpy.array ([g0,g1])
30. for j in range (N-1):
31. #x[j+1]=euler(x[j],time[j],dt, Ecu)
32. #x[j+1]=euler_richar(x[j],time[j],dt,Ec
    u)
33.   x[j+1]=runge_kutta(x[j],time[j],dt,Ec
    u)
34. datos_x=[x[j,0] for j in range (N)]
35. datos_v=[x[j,1] for j in range (N)]
36. plt.subplot(2,1,1)
37. plt.plot(time, datos_x, 'ro--')
38. plt.xlabel('tiempo(s)')
39. plt.ylabel('Posición(m)')
40. plt.grid()
41. plt.subplot(2,1,2)
42. plt.plot(time, datos_v, 'bo--')
43. plt.xlabel('tiempo (s)')
44. plt.ylabel('Velocidad(m/s)')
45. plt.grid()
46. plt.show()
```

Para el caso presentado se muestra la simulación del movimiento de caída libre. En las líneas 22 a 28 se definen las variables iniciales de la simulación: el número de pasos, el tiempo total, los intervalos de tiempo, la posición inicial, la velocidad inicial y la gravedad.

En la línea 29 se define la matriz de resultados, que para el caso será de 30 valores, dos columnas, y donde los valores iniciales de cada columna corresponden a la posición y la velocidad inicial. En la línea 32 se define la función que contiene la ecuación diferencial que se va a resolver, en la línea 36 se aplican los algoritmos de *Euler*, *Euler Richardson* y *Runge Kutta*, para finalizar en las líneas 40 a 52, con el procedimiento de hacer los gráficos correspondientes. Al terminar el procedimiento se obtienen los resultados de la Figura 1.

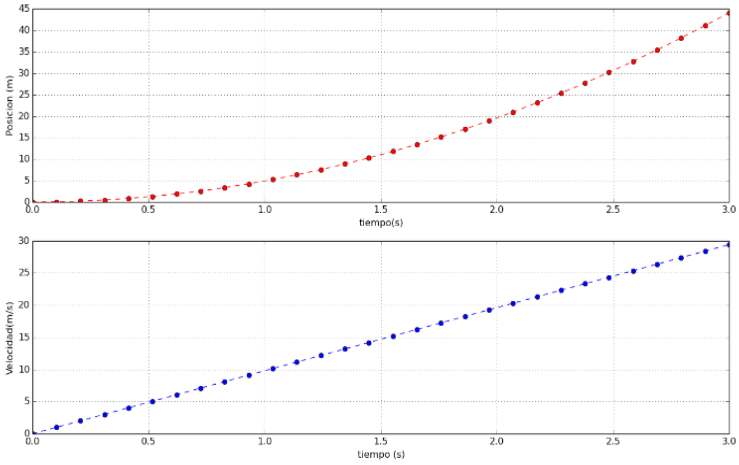

Figura 1. Ejemplo de presentación de resultados de simulación del experimento de caída libre usando Python. Arriba: posición (m) Vs tiempo(s). Abajo: velocidad (m/s) Vs tiempo(s).

2.2 Simulación mediante Interfaz gráfica de usuario, usando Qt Designer

Utilizando los algoritmos explicados en la sección anterior, se desarrolla una interfaz en Python – Qt4, empleando la herramienta Qt Designer. En la Figura 2, se aprecia el diseño de la interfaz gráfica, en donde se encuentran los campos para ingresar información relevante para los experimentos (constantes y condiciones iniciales), una ventana para la selección de la simulación que se va a realizar, una tabla de resultados y un botón de inicio de simulación. Cada experimento tiene sus propios parámetros, es decir, en *Caída Libre* los campos por ingresar son velocidad y altura, en el *Movimiento Rectilíneo Uniforme* son velocidad y tiempo, en el *Movimiento Rectilíneo Uniformemente Variado* son velocidad, aceleración y tiempo, y en el *Movimiento Parabólico* los campos son velocidad y theta (ángulo de lanzamiento). Estos campos se activan al seleccionar el tipo de movimiento.

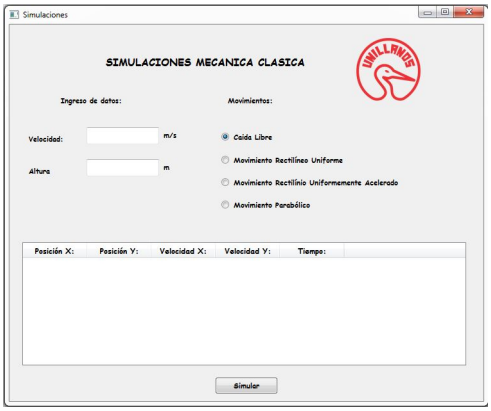

Figura 2. Interfaz gráfica de usuario. Python - Qt4.

Al realizar la simulación se pueden observar los resultados numéricos y un gráfico que muestra la ubicación de la partícula en cada instante de tiempo simulado. En la Figura 3 se aprecia la tabla de datos obtenida en el caso del movimiento parabólico con los campos de velocidad=30 m/s y ángulo de lanzamiento, theta= 45°. En la Figura 4 se visualizan dos gráficas obtenidas en la simulación correspondientes a la *Posición_Y(m) Vs. Posición_X(m)* y *Posición_Y(m) Vs t(s).*

Al finalizar cada simulación, los datos se almacenan con información de fecha y hora en una hoja de cálculo, para posteriores análisis y procedimientos (Figura 4).

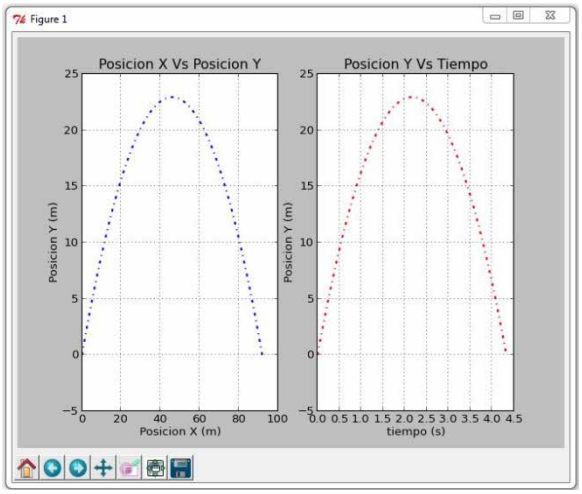

Figura 3. Gráficos de la simulación. *Izquierda: Posición_Y(m) Vs. Posición_X(m)* y *derecha: Posición_Y(m) Vs t(s).*

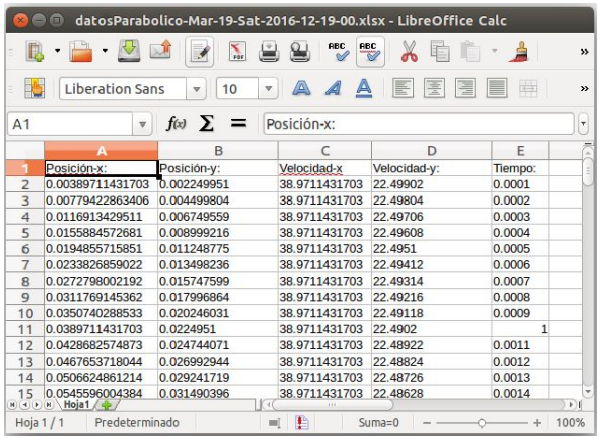

Figura 4. Información obtenida en el experimento visualizada mediante hoja de cálculo.

2.3 Simulación mediante Visual Python y Tkinter

Finalmente, usando la librería Visual Python se desarrollan entornos de simulación de mejor calidad, en donde se hacen gráficos y se estudian movimientos con interactividad en tres dimensiones. Al hacer simulaciones mediante esta librería de Python, se trabajan aspectos como el fondo, el tamaño y los colores de las formas. En la Figura 5 se aprecia la simulación del movimiento rectilíneo uniforme (MRU).

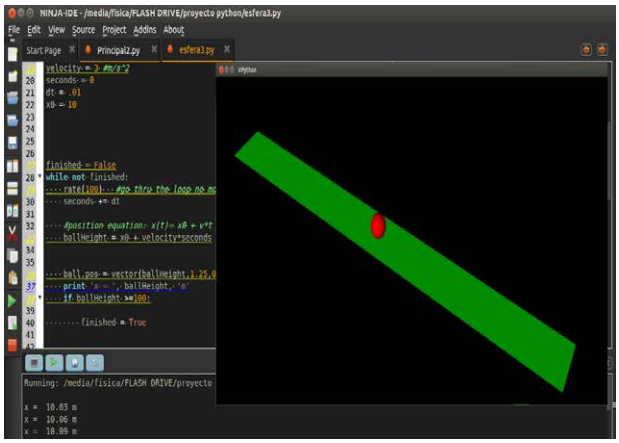

Figura 5. Simulación del Movimiento Rectilíneo Uniforme mediante Vpython.

El código empleado para lograr la simulación de la Figura 5 se muestra a continuación:

```
1.  import visual
2.  from visual import *
3.  scene.width = 700
4.  scene.height = 500
5.  scene.range = (100,100,100)
6.  scene.center = (50,20,0)
7.  ball=sphere(pos=(0,2,0),radius=2, c
    olor = color.red)
8.  ground=box(pos=(50,-
    0.5,0),size=(100,0,10), color = col
    or.green)
9.  velocity = 30
10. seconds = 0
11. dt = 0.01
12. x0 = 10
13. finished = False
14. while not finished:
15.  rate(100)
16.  seconds += dt
17.  ballHeight = x0 + velocity*seconds
18.  ball.pos=vector(ballHeight,1.25,0)
19.  if ballHeight >=100:
20.          finished = True
```

En las líneas 1 y 2 se hace el llamado a la librería Vpython, mediante la sentencia *import visual*, y el llamado de todas las funciones referentes (*from visual import **). En las líneas 3 a la 6 se define la escena de trabajo, iniciando con ancho y alto, rango y centro. En las líneas 7 y 8 se definen las características de los objetos de la simulación, que para este caso corresponden a una esfera de color rojo y un plano horizontal de color

verde. Las líneas 9 a 12 definen las condiciones iniciales de la simulación, para finalmente con las líneas 14 a 20 ejecutar la simulación del evento físico.

3. RESULTADOS

Para complementar los resultados de las simulaciones y mejorar la interactividad con el usuario, se desarrolló con la ayuda de la librería *Tkinter*, que es compatible con *VPython,* un entorno gráfico que se pueda manipular. Se obtiene una aplicación que simula los movimientos planteados en el trabajo de grupo de estudio: movimiento rectilíneo uniforme (MRU), rectilíneo uniformemente variado (MRUV), caída libre y parabólico, permitiendo graficar en ventanas emergentes los parámetros fundamentales respecto al tiempo y almacenar los datos que se obtienen en la simulación en un archivo de texto plano.

En la Figura 5, se aprecia la simulación del movimiento de movimiento rectilíneo uniforme (MRU), en la Figura 6, la simulación del movimiento uniformemente variado (MRUV), en la Figura 7 caída libre, y en la Figura 8 la del movimiento parabólico. Se puede seleccionar el tipo de simulación que se va a realizar, colocar los parámetros iniciales del movimiento, mostrar los gráficos correspondientes. Se resalta que la aplicación permite interactuar directamente con la ubicación en la que se desea observar el comportamiento.

Cada simulación tiene la posibilidad de visualizar el logo de la universidad, los tipos de simulaciones y un botón de salida. Al oprimir en cualquiera de estos botones, se ingresará a una segunda ventana (inferior derecha), la cual tiene una caja de texto para poder ingresar los datos y el tipo de dato que se va a ingresar; dos botones, uno es para salir de la simulación y terminar el programa, el segundo botón es el de iniciar la simulación y se desplegarán los tres *frames*, compuestos por dos ventanas centrales, las cuales contienen gráficas cartesianas posición versus tiempo, velocidad versus tiempo, posición_y versus posición_x, dependiendo del fenómeno físico; el último *frame* (ventana izquierda) contiene la simulación del fenómeno físico.

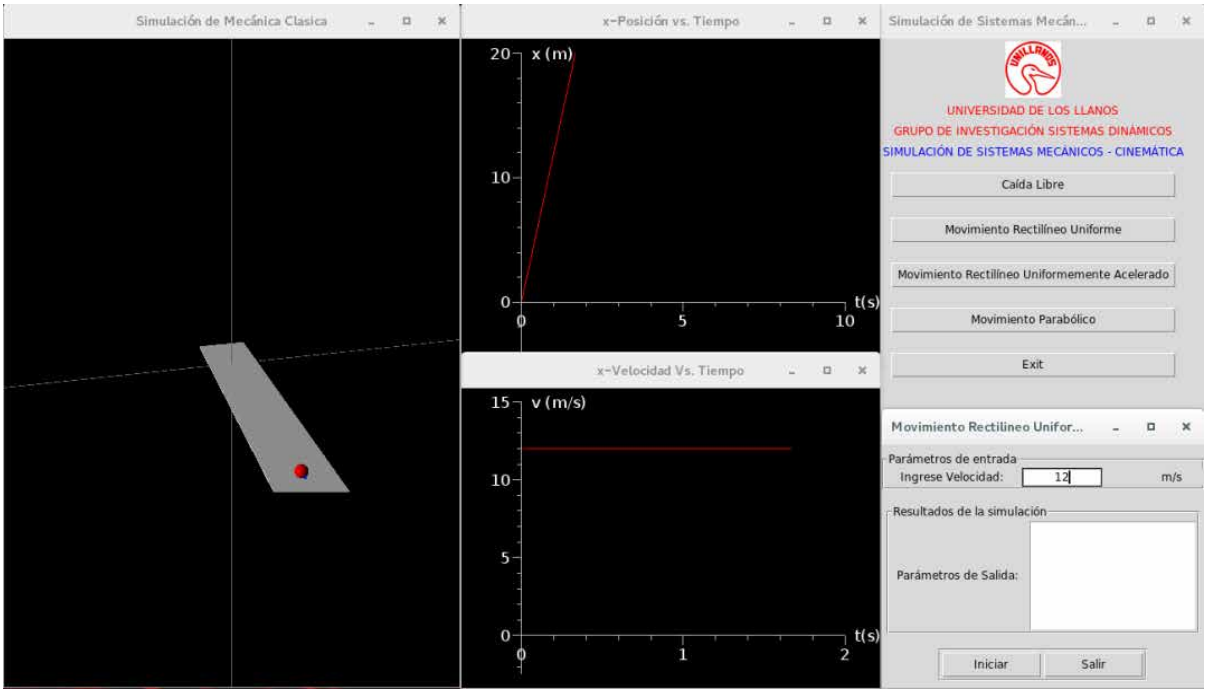

Figura 6. Aplicación desarrollada. Simulación del movimiento rectilíneo uniforme (MRU).

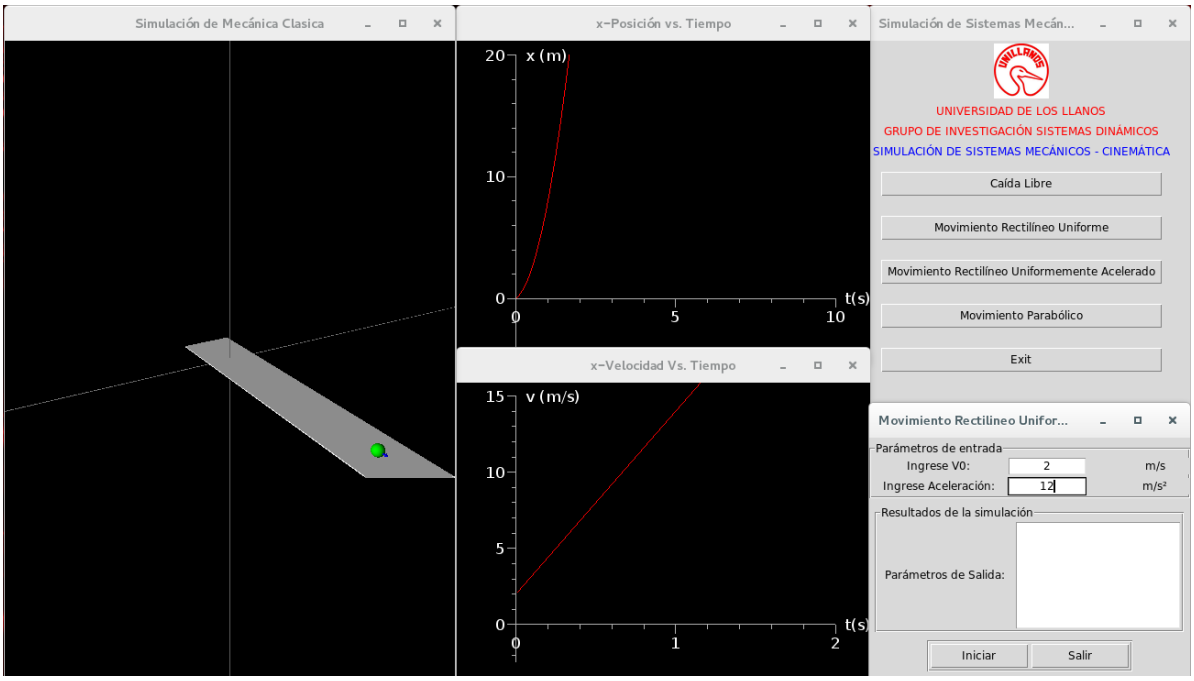

Figura 7. Aplicación desarrollada. Simulación del movimiento rectilíneo uniformemente variado (MRUV).

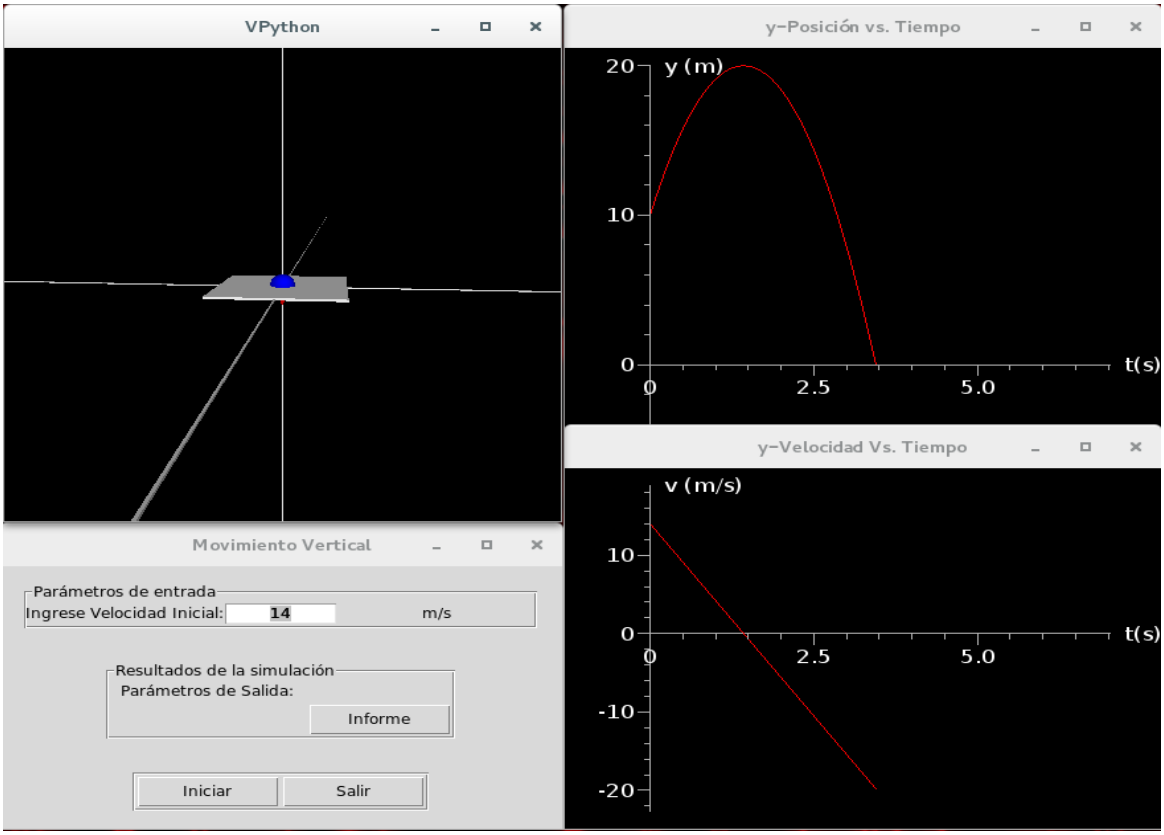

Figura 8. Aplicación desarrollada. Simulación de caída libre.

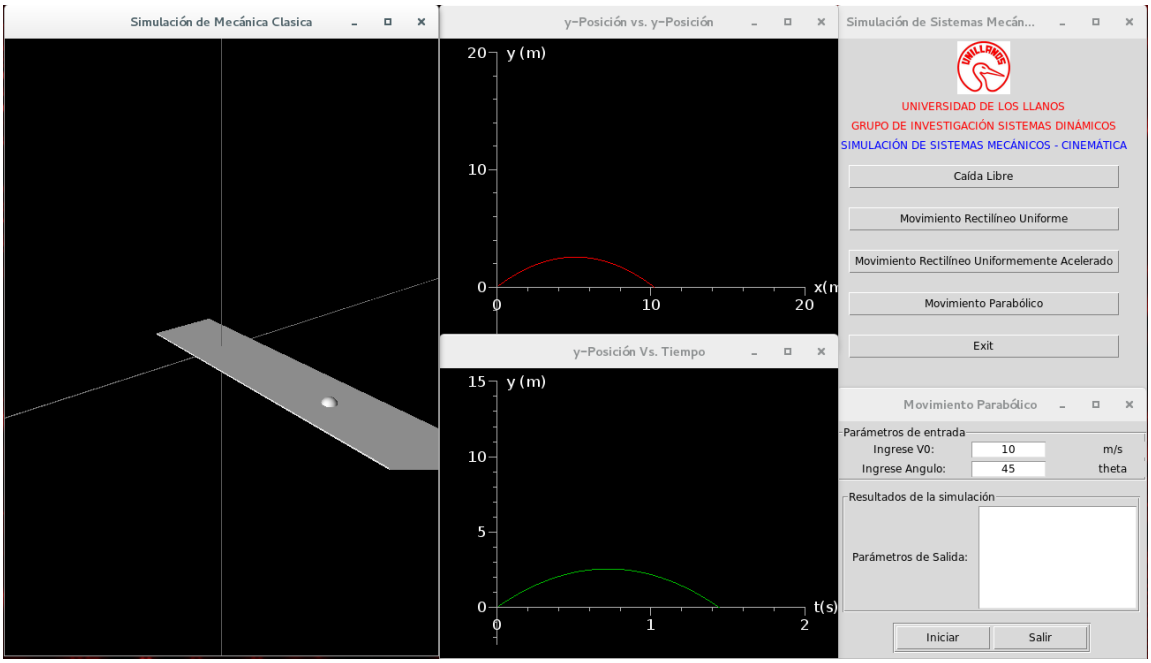

Figura 9. Aplicación desarrollada. Simulación movimiento parabólico.

4. CONCLUSIONES

Cuando se utiliza la simulación para la enseñanza de la cinemática, los estudiantes pueden entender el comportamiento de una partícula en movimiento (manipulación y visualización mediante una interfaz gráfica) y el principio matemático de las ecuaciones que gobiernan estos fenómenos (desarrollo de algoritmos computacionales).

Python y Tkinter son herramientas poderosas para el desarrollo de aplicaciones en *software*, que adicionando librerías como Visual Python y QtDesigner permiten lograr aplicaciones útiles para la enseñanza de la física. Usando estas herramientas se complementa el aprendizaje de la simulación de procesos físicos en el aula, de forma tal que existe gran interactividad con el usuario para obtener resultados, además de permitir el aprendizaje de los algoritmos utilizados en cada etapa del procesamiento.

Mediante el uso de las herramientas computacionales, se ha logrado que el procedimiento en el desarrollo de aplicaciones de la física computacional en la Universidad de los Llanos tenga fundamentos básicos para el impulso de aplicaciones más avanzadas, haciendo que el estudiante de ingeniería muestre mayor interés en la realización de simulaciones en diferentes tópicos de la física, en el grupo de investigación Sistemas Dinámicos.

5. AGRADECIMIENTOS

Agradecemos a la Dirección General de Investigaciones y al grupo de investigación Sistemas Dinámicos de la Facultad de Ciencias Básicas e Ingeniería de la Universidad de los Llanos, por su apoyo en el desarrollo del proyecto de investigación código FCBI-13-2013.

6. REFERENCIAS

Bransford, J., Brown, A. & Cocking, R. (2004). *How People Learn: Brain, Mind, Experience, and School.* Washington DC: National Academy Press. DOI: 10.17226/9853.

Bustacara, C. J. (2010). Evaluación computacional para calcular los polinomios de Legendre de primera clase. *Avances en Sistemas e Informática, 7*(2), 131-138.

Chickering, A. & Ehrmann, S. (1996). Implementing the Seven Principles: Technology as Lever. *American Association for Higuer Education*, 3-6.

Egas, M. E. (2014). *Simulación computacional de la trayectoria de electrones de incepción en un tramo corto de aire.* Quito, Ecuador: Editorial Quito.

García, A., Martín, J. A. & Gutiérrez, M. T. (2010). Modelo computacional para la formación de clases de equivalencia. *International Journal of Psychology and Psychological Therapy, 10*(1), 163-176.

Landau, R., Páez, M. J. & Bordeianu, C. (2010). *A survey of Computational Physics.* Oxford: Princeton University Press.

López, O. R., Narváez, C. A. & Garzón, D. A. (2012). Modelos computacionales del comportamiento del cartílago articular. *Revista Cubana de Investigaciones Biomédicas, 31*(2), 373-385.

Overholt, K. (2010). Numerical Pyromaniacs: The Use of Python in Fire Research. *9th Python in Science Conference SCIPY.*

Panou, T. (2008). Management of Learning Ways: a Radiographer's prospective. *Athens T.E.I MIR*.

Sener, J. (1997). Constructivism: Asynchronous Learning Networks. *ALN Magazine, 1*, 1.

Vargas, W. & Murcia, J. C. (2005). Distribución de fuerzas en medios granulares no cohesivos: observaciones experimentales y computacionales. *Ciencia e Ingeniería Neogranadina,* (15), 138-150.

Vilchez, A., Marzocchi, V., Beldoménico, H. & Vanzetti, N. (2013). Uso de software Libre para un portal de compuestos orgánicos persistentes. *10mas Jornadas Argentinas de software Libre.* Buenos Aires.

Yasar, O. (2006). A computational technology approach to education. *Computing in Science and Engineering, 8*(3), 76-81.

Yasar, O. & Landau, R. (2003). Elements of computational Science and engineering Education. *Society for Industrial and Applied Mathematics, 45*(4), 787-805. DOI: 10.1137/S0036144502408075

Zamarro, J., Molina, G. & Núñez, M. (2004). Teaching Physics Modelling with Graphic Simulations Tools. *HSCI.*