Física Computacional: una propuesta educativa

por J. F. Rojas-Rodríguez, M. A. Morales-Sánchez, A. Rangel-Huerta

Facultad de Ciencias Físico Matemáticias. B. Universidad Autónoma de Puebla
18 sur y Av. San Claudio, C. U., Col. San Manuel, Puebla, Pue. C. P. 72570

Correo-e: frojas@fcfm.buap.mx

Resumen

A partir de características centrales del lenguaje de programación python se muestra cómo puede convertirse éste en una herramienta didáctica importante en la formación de un profesional de la Física. Se ha aunado esta idea a la de Física Computacional, que se propone con objetivos y contenidos diferentes a los cursos tradicionales de Programación y de Métodos Numéricos. Se propone partir, como estrategia, de los modelos más simples de la Física y de su interpretación aproximada, numérica, relacionada todavía con las ideas pre-diferenciales. Ahí mismo se aprovechan los recursos de cómputo y las posibilidades que ofrece python para generar imágenes visuales o animaciones (ya sea a partir de datos o bien creando animaciones en tiempo real). Se muestra con ejemplos la posibilidad de ir incrementando el peso conceptual de los problemas manteniendo nexos con los problemas anteriores ya sea por la física misma, los métodos utilizados o por analogía.

PACS: 01.40.gb, 01.50.Lc

1 Introducción

Desde hace ya unas décadas, se ha convertido en una cuestión importante para las ciencias el empleo de recursos computacionales. Aspectos de la actividad científica tales como la edición de material didáctico, reportes o artículos de investigación y divulgación los requieren, pero también hay usos tales como la evaluación de integrales que no tienen solución analítica o cuya solución es muy complicada, o bien la solución de ecuaciones o de sistemas de ecuaciones algebraicas o diferenciales, el análisis estadístico de datos, la simulación de procesos, la evaluación de modelos físicos, etc. [8]

La experiencia ha mostrado una cuestión importante en los cursos tradicionales de Programación y de Métodos Numéricos y es, precisamente, el hecho de que los estudiantes realmente no aprenden a programar, ni aprenden a utilizar los recursos que ofrece un equipo de cómputo en términos de su actividad profesional. Este hecho está aunado a otro que es real: difícilmente en su actividad científica van a utilizar las cosas que se enseñan en un curso tradicional de Métodos Numéricos. De esta suerte ocurre que, normalmente,

cuando enfrentan su trabajo de tesis u otros trabajos académicos, la parte computacional se vuelve una especie de lastre pesado. Las posibilidades hoy son más, y más amplias [4].

Parte del problema, creemos, proviene del hecho de que se intenta que ellos aprendan a utilizar una herramienta -la computadora- y muchos de sus elementos de software, pero nunca con objetivos sobre los cuales se supone que la van a utilizar: los problemas de la física y/o de otras áreas. Una cuestión que puede funcionar como un elemento motivador es el planteamiento de un reto (un problema) que habrá que aprender a resolver con una herramienta que posee alguna «habilidad» desconocida y, en este proceso las técnicas y métodos irán apareciendo como elementos necesarios para llegar a la solución requerida.

De una parte el concepto de Física Computacional, de otra la utilización de lenguajes de programación accesibles, robustos y con menos estructura y restricciones que los convencionales C, C++ o Fortran[1]. Esta es la propuesta, y su enfoque pedagógico se basa en la resolución práctica de problemas, de la física y de otras áreas[2], simples al inicio, y que se van complicando y haciendo más realistas conforme se avanza. En este caso «simples» se refiere más bien a la posibilidad de implementar una solución, o una simulación del problema, de manera computacionalmente sencilla y, al mismo tiempo, que no requiera de conocimientos profundos o avanzados de la física.

La resolución de problemas en sesiones tipo taller permite, por otro lado, incrementar la posibilidad de resolver problemas de manera colectiva con la facilidad adicional de poder implementar la solución de forma inmediata en la computadora. Aquí cobra mucha importancia el lenguaje y las herramientas de cómputo en general: deben ser tales que los estudiantes puedan implementar de la forma más tranparente y directa el algoritmo que acaban de construir. Al finalizar el curso-taller, ellos serán capaces de proponer y plantear sus propios problemas a partir de inquietudes personales, y de buscar los elementos que les permitan, por lo menos, acercarse a una posible solución a su inquietud. Obviamente esta parte tiene que ser promovida por el que guía el aprendizaje.

Por el momento esperamos que este texto sirva como invitación o motivación y, por lo menos, como una pequeña guía de las cosas que se pueden hacerse en el salón de clase o en la computadora personal y que pueden llevar, a quien siga el camino, tan lejos como quiera caminar.

1. En general es un hehco que cuando una persona puede programar y resolver problemas en cualquier lenguaje, entonces bastará con una guía mínima o un manual para que pueda hacerlo en cualquier otro. Lo importante es que los estudiantes desarrollen su habilidad para unir los conceptos, la matemática y sus técnicas para implementar algoritmos y soluciones.

2. Las técnicas y métodos para resolver numéricamente, por ejemplo, sistemas de ecuaciones diferenciales, son universales, así que se pueden aplicar a problemas de diferentes áreas del conocimiento.

A partir de la sección 3 se desarrollan ejemplos orientados a la discusión de los algoritmos con los cuales resolver, simular o modelar algunos fenómenos típicos de la física. En la sección se inicia con la cinemática de una partícula en una y dos dimensiones para continuar con la dinámica de un oscilador amortiguado (el caso armónico es un caso particular que se obtiene muy fácilmente de ahí) en el espacio fase. En la sección 4 hay un salto conceptual: ahora se trata de sistemas de muchas partículas y, de inicio, se discute el caso de partículas «libres» dentro de un recipiente (el gas ideal). Después el caso de movimiento Browniano en donde aparece la idea de fuerzas aleatorias. Por último, con el objeto de introducir otros módulos de `python`, se discute el gas de esferas duras: ahora se tiene un gas, como antes, pero las partículas tienen volumen y sus interacciones se dan a través de colisiones. La última sección presenta, como ejemplo, una lista de preguntas de control que pueden hacerse a los alumnos del curso. Estas preguntas están dirigidas, sobre todo, a conceptos algorítmicos, pero también a la física computacional, a cómo interpretar, modelar o calcular ciertas cantidades de interés de un sistema.

2 Características de `python`

Antes de enumerar un conjunto de características del lenguaje `python` que lo hacen idóneo para algunas tareas en términos de docencia, debo comentar que la elección del lenguaje está basada sobre todo en la idea de que los cursos de física computacional deben ser más cursos de discusión de la física, de las formas de implementar soluciones, de visualizar dinámicas, de construir modelos y simular eventos. Un lenguaje de programación que nos permita esto siempre será bienvenido como un buen recurso didáctico [1].

Se pueden enlistar muchas características del lenguaje de programación `python`. De manera particular se enumeran algunas que permiten a los estudiantes comenzar a programar de manera inmediata y sencilla [2, 9]:

- Se trata de un lenguaje interpretado[3], de modo que se puede usar de modo interactivo o bien a través de la construcción de un *script*.
- Cualquier script puede ser utilizado como un módulo, es decir, una función u objeto que ya se construyó en algún archivo puede ser importada desde cualquier otro.
- El lenguaje está orientado a objetos, los cuales se implementan de manera muy sencilla.

3. Esto puede ser una desventaja en términos de la velocidad de ejecución. En términos prácticos, sin embargo, el tiempo de ejecución se puede compensar de dos formas: por un lado el tiempo de programación se reduce muchísimo y, por otro `python` permite incorporar rutinas escritas en `fortran` y/o `C++`. De hecho la librería `numpy` utiliza las conocidas librerías `lapack` y `blas` que están escritas en `fortran`.

- Cuenta con librerías o módulos para cálculo numérico (Numeric, numpy) y aplicaciones científicas (scipy) así como para hacer aplicaciones interactivas así como animaciones gráficas (Tkinter, VPython, wxPython) cuya sintaxis es simple (como la del lenguaje en general).
- No requiere declaración de variables: se declaran cuando se les asigna un valor que, obviamente, puede cambiar durante la ejecución.
- Los apuntadores son algo natural en `python` y están asociados a los datos que tienen estructura interna (listas o arreglos).
- Las listas, son objetos muy flexibles que pueden contener simultáneamente cualquier tipo de dato escalar, nombres de funciones (apuntadores), otras listas, cadenas de caracteres, etc. Las librerías de uso numérico, como `numpy`, utilizan las listas y muchas de sus propiedades.
- Se trata de software libre (*freeware*) y existe para todos los sistemas operativos.
- Existen en internet una gran cantidad de módulos, librerías, aplicaciones, documentación, etc.
- Se pueden hacer funciones en C y Fortran que se ligan con `python`

Existen ejemplos muy simples que aparcen en cualquier manual o libro de programación: el primer programa que el usuario haría y que simplemente escribe la frase «Hola. mundo», puede escribirse de mucha maneras de acuerdo al lenguaje utilizado, pero en python se escribe tan simple como:

```
print 'Hola, mundo'
```

del mismo modo que la evaluación de la serie $\sum_{k=0}^{n} \frac{1}{2^k}$ se puede hacer con[4]

```
sum([0.5**k for k in range(n)]).
```

3 Algunas experiencias de Mecánica con `python`

Para un estudiante medio de cualquier licenciatura en física se puede asociar cada uno de los problemas propuestos aquí a diferentes grados de avance dentro de los primeros cursos de mecánica, sin embargo, cada uno de los problemas aquí planteados está diseñado para avanzar, en la medida que lo permitan los conocimientos y experiencia del estudiante, hacia tópicos conceptualmente más profundos y, al mismo tiempo, de aplicaciones a la solución de problemas cada vez más relacionados con otras áreas del conocimiento y del quehacer científico.

4. Aunque este tipo de aplicación no la hemos visto en ningún manual todavía.

3.1 Cinemática en una dimensión

Se puede comenzar por suponer, como en casi cualquier primer curso de Física, un cuerpo que se desplaza de manera que recorre distancias iguales en tiempos iguales. De esta manera se tiene que, en una dimensión,

$$v = \frac{\Delta x}{\Delta t} \tag{1}$$

de modo que siempre se pueden tomar incrementos iguales de tiempo Δt que corresponden a distancias recorridas, Δx, iguales. Se trata de que v sea constante. Si esto último no ocurre, obviamente, ya no se trata de un movimiento rectilíneo y uniforme. De cualquier forma siempre se puede escribir la expresión aproximada

$$\Delta x = v\,\Delta t \tag{2}$$

o bien, de aquí,

$$x_n - x_0 = n\Delta x$$

de manera que

$$x_n = x_0 + n\Delta x = x_0 + vn\Delta t \tag{3}$$

que nos da la posición $x_n = x(t_n)$ en el instante $t_n = n\Delta t$ con $n = 0, 1, \ldots$ para cierta velocidad v constante[5]. Equivalentemente, si se hace una partición uniforme del intervalo $[x_0, x_f]$ con subintervalos de tamaño Δx, la segunda forma de la expresión (3) nos dará los valores sucesivos de la coordenada en la forma discreta, x_n.

Se tienen ya los elementos para generar una lista de pares ordenados que nos permitan construir la gráfica $x - t$ del objeto:

- en python:

```
# grafica de movimiento rectilineo uniforme

x0 = 34.5
v = 143.2
delta_t = 0.01
puntos = 5000

# un par de arreglos
t = [n*delta_t for n in range(puntos)]
x = [x0+n*delta_t*v for n in range(puntos)]

# escribir en archivo ‘‘datosv.dat’’
```

5. Y entonces el profesor se pone a disertar, o bien pone a los estudiantes a rediscutir y recordar, sobre la relación que hay entre esta expresión y una ecuación *lineal* $y = ax + b$, la pendiente y todos esos conceptos.

```
f = open('datosv.dat','w')
for i in range(puntos):
   print > f, t[i], x[i]
f.close()
```

en donde `range(puntos)` es una lista de valores enteros desde cero a `puntos-1`.

Dado que la expresión (1) siempre es válida si $\Delta t \to 0$ (o, en nuestro caso, si $\Delta t \ll 1$) la cantidad que representa la velocidad v podría no ser una constante. De hecho podría ser una función, por ejemplo, del tiempo, de la posición, etc.

Primero, la expresión (2) se modifica por el hecho de que v no es constante, de modo que

$$\Delta x_n = v_n \, n\Delta t \tag{4}$$

si Δt se mantiene constante, como se acostumbra. El subíndice n nos indica que el valor del incremento en x, o distancia recorrida Δx_n, ahora depende del momento en que se observa $t_n = n\Delta t$, de la misma manera que lo hace la velocidad v_n. Si $\Delta x_n = x_{n+1} - x_n$, entonces la expresión (3) no se modificaría sustancialmente. Si definimos $x_{n+1} = x(t_{n+1})$ se tiene que[6]

$$x_{n+1} = x_n + \Delta t \cdot v_n, \tag{5}$$

es decir, que la nueva posición depende de la anterior y de la velocidad $v_n = v(t_n)$ que corresponde al intervalo [17]. El programa quedaría de la forma siguiente

- en python:

```
# ahora con velocidad variable

x0 = 34.5
delta_t = 0.01
puntos = 5000

# se define la funcion velocidad
def v(t):
   return 3.4*t**2

# un par de arreglos o listas
t = [n*delta_t for n in range(puntos)]
x = [x0 for n in range(puntos)]
for n in range(puntos-1):
   x[n+1] = x[n] + delta_t*v(t[n])  # la fórmula (5)
```

6. Aquí uno puede platicar sobre el Método que implementó Euler para resolver de modo aproximado una ecuación diferencial de la forma $dy/dx = F(x, y)$ y de cómo esa aproximación, al igual que otras, se puede obtener de una expansión en serie de Taylor. Finalmente la expresión (5) es la expresión de la fórmula de Euler simple. El método de Euler, aunque es muy sencillo, permite ver las cualidades de la dinámica de los sistemas.

```
# escribir en archivo ‘‘datosvar.dat’’
f = open('datosvar.dat','w')
for i in range(puntos):
   print » f, t[i], x[i]
f.close()
```

El archivo de datos puede graficarse de forma sencilla con `gnuplot` u otros programas de graficado. En la figura 1 se encuentran superpuestas las dos gráficas de los ejemplos.

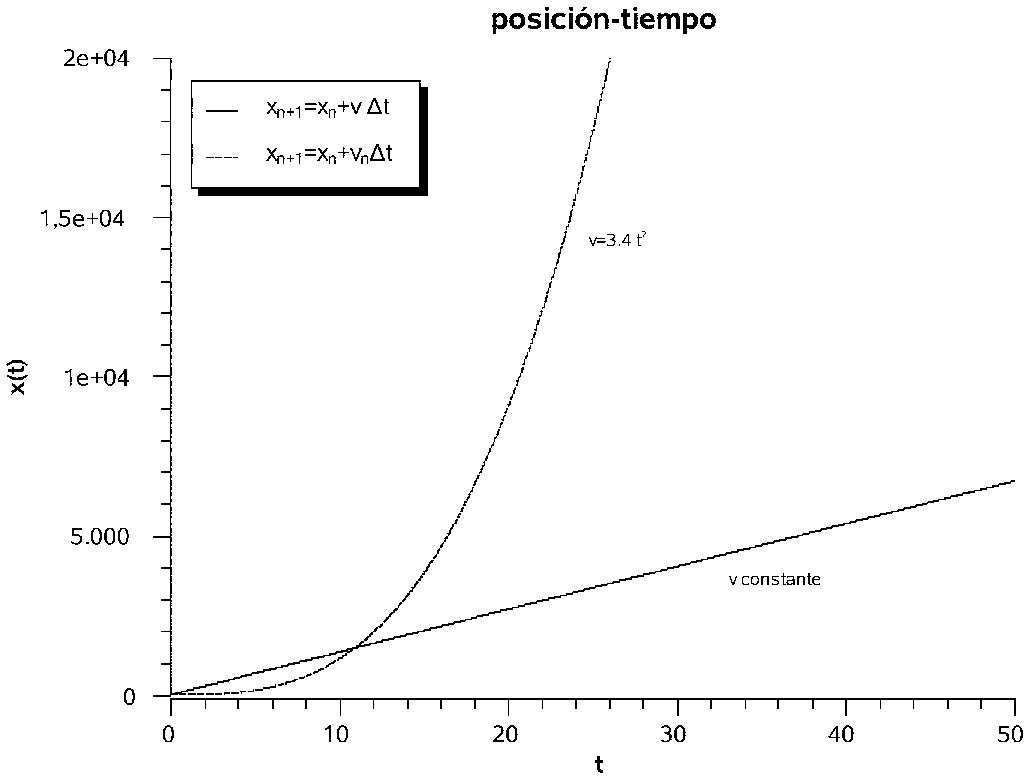

Figura 1. Posición como función del tiempo. Los dos casos.

3.2 Cinemática en dos dimensiones

El caso bidimensional se extiende de manera muy simple: una caraterística de las funciones en python es el hecho de que los datos no tienen tipo predefinido y eso hace que se conviertan en objetos muy flexibles. Si se piensa en las ecuaciones paramétricas, por ejemplo de algún objeto cuyas velocidades se conocen, digamos

$$\boldsymbol{v}(t) = (v_x(t), v_y(t)) = (-\operatorname{sen} t, \cos t), \tag{6}$$

tenemos un par de ecuaciones que deben resolverse de la forma (5). Solamente hay que pensar que se requieren la dos soluciones en función del tiempo, $x(t)$ e $y(t)$, y la trayectoria $x - y$. Así que se trata de utilizar la estructura *lista* de `python` (una *lista* no funciona como vector necesariamente[7])

- en python:

```
# ahora con velocidad variable
# y en dos dimensiones

#funciones matematicas:
from math import *

# condición inicial (vector)
r0 = [34.5, 19.3]
dt = 0.01
puntos = 5000

# se define la funcion velocidad 2D
# con delta t incluido
def v(t):
   return [-sin(t),cos(t)]

# un par de arreglos: x0 es un vector
t = [n*dt for n in range(puntos)]
r = [r0 for n in range(puntos)]
for n in range(puntos-1):
   r[n+1][0] = r[n][0] + dt*v(t[n])[0] # la fórmula (5)
   r[n+1][1] = r[n][1] + dt*v(t[n])[1] # otra vez
# escribir en archivo ‘‘datosX.dat’’
# t, x, y
f = open('datosvar.dat','w')
for i in range(puntos):
   print » f, t[i], r[i][0], r[i][1]
f.close()
```

En este caso `r0[0]` representa la coordenada x_0 y `r0[1]` la coordenada y_0, de la misma manera que `r[i][0]` y `r[i][1]` son las coordenadas x_i, y_i en el mismo sentido de la expresión (5).

7. Una lista tiene la forma `a=[0,1,2,'hola',['A',45]]`. Esto no puede ser un vector: conceptualmente es otra cosa, pero puede emplearse, con cuidado, como tal [REFS].

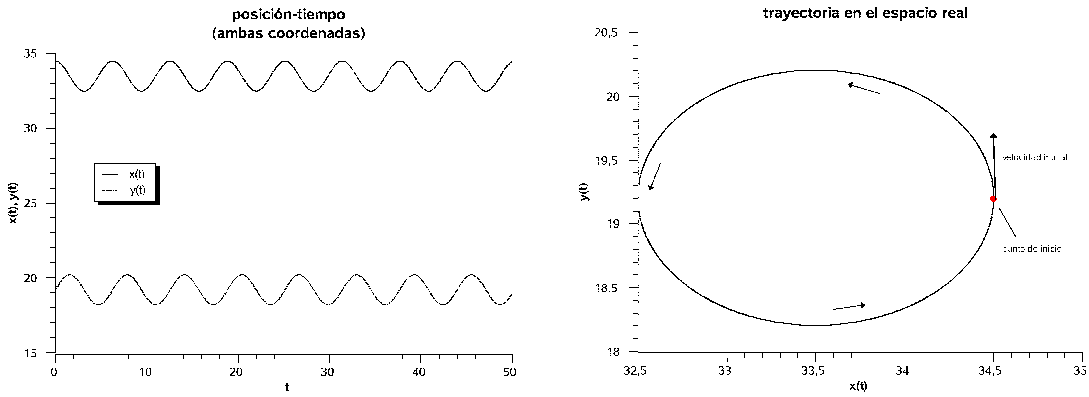

Figura 2. Las soluciones $x(t), y(t)$ y la trayectoria.

3.3 Dinámica en una dimensión

El problema del oscilador armónico clásico, unidimensional, a partir de la ecuación de movimiento, es un problema análogo al anterior en lo referente a las formas de resolverlo, lo único que cambia es el enfoque del problema: tenemos una trayectoria pero en el *espacio fase* y tenemos dos variables que no son las coordenadas[8]: son la posición y la velocidad.

En Mecánica Teórica se aprende, entre otras cosas, que hay dos formas de resolver un problema, o bien dos formas de plantear formalmente las ecuaciones a resolver: se tiene una ecuación diferencial de segundo orden, o se tienen dos ecuaciones diferenciales de primer orden [11]. La ecuación de movimiento del oscilador armónico con fricción[9] es

$$\ddot{x} = -\frac{k}{m}x - a v^{b} \tag{7}$$

que se puede transformar, pensando en un oscilador armónico universal que se pueda tomar como patrón, en la ecuación

$$\ddot{x} = -x - a v^{b} \tag{8}$$

y si además definimos un par de variables dinámicas $x_0 = x$ y $x_1 = \dot{x}$ tendremos un par de ecuaciones diferenciales de primer orden[10]

$$\dot{x_0} = x_1 \tag{9}$$

8. En el enfoque que veremos aquí, de Sistemas Dinámicos, las variables pueden ser cualquier magnitud medible del sistema. Aquí nos restringimos a problemas de mecánica, sin embargo, la formulación permite extenderse a problemas del tipo depredador-presa o dinámica de reacciones químicas (que es aplicable, por ejemplo, a problemas de polución atmosférica, crecimiento de tumores, dinámica cardiaca o de la respiración, ecología, modelos económicos, etc.)

9. Basta hacer $a = 0$ para tener el caso del oscilador armónico simple.

y

$$\dot{x}_1 = -x_0 - a\,x_1^b, \tag{10}$$

de modo que ahora tenemos dos variables dinámicas y un par de ecuaciones de la forma genérica

$$\begin{aligned} \dot{x}_0 &= F_0(x_0, x_1) \\ \dot{x}_1 &= F_1(x_0, x_1), \end{aligned}$$

que es representativa de la dinámica de muchísimos sistemas no necesariamente relacionados con la física: se trata del enfoque de los Sistemas Dinámicos [16]. El método de Euler para este caso tiene la forma, para $k = 0, 1$

$$x_{t+1,k} = x_{t,k} + dt \cdot F_k(x_{t,0}, x_{t,1}). \tag{11}$$

El interés consiste, entonces, en resolver las ecuaciones simultáneas (9) y (10). Hay que notar que el problema es análogo al de emplear el método de Euler (5) para dos variables, tal como se hizo en el programa para resolver el oscilador armónico en dos dimensiones.

- en python sería:

```
# dinámica del oscilador
# armónico simple

# condición inicial (vector de
# posicion y velocidad iniciales)
x0 = [4.5, 1.3]
dt = 0.01
puntos = 1000

# una funcion de friccion...
a, b = 0.6, 1.2
def f(v):
   return a*abs(v)**b

# se definen las funciones
# del lado derecho
def F0(X):
   return X[1]
def F1(X):
   return -X[0]-f(X[1])
```

10. Nótese aquí que estamos hablando de dos variables que definen el espacio fase del sistema. En general este tópico se presta, dependiendo del nivel de la exposición, para rediscutir o recordar las formulaciones de Lagrange y de Hamilton. En este último caso se habla Sistemas Hamiltonianos y la discusión puede continuar hacia temas como el caos, la ergodicidad, etc. [16, 14]

```
# x es una lista (vector) de pares
# ya que x0 es un 'par' (vector):
x = [x0 for n in range(puntos)]
for t in range(puntos-1):
   x[t+1][0] = x[t][0] + dt*F0(x[t]) # la fórmula (5)
   x[t+1][1] = x[t][1] + dt*F1(x[t]) # otra vez
# escribir en archivo ‘‘datosF.dat’’
# t, x, v
f = open('datosF.dat','w')
for i in range(puntos):
   print » f, dt*i, x[i][0], x[i][1]
f.close()
```

de este modo al graficar las columnas 2 con 1 se tiene la gráfica posición-tiempo. De la misma forma, usando las columnas 3 con 1, la de velocidad-tiempo. La imagen fase (*phase portrait*) se obtiene al graficar la columna 3 contra la 2.

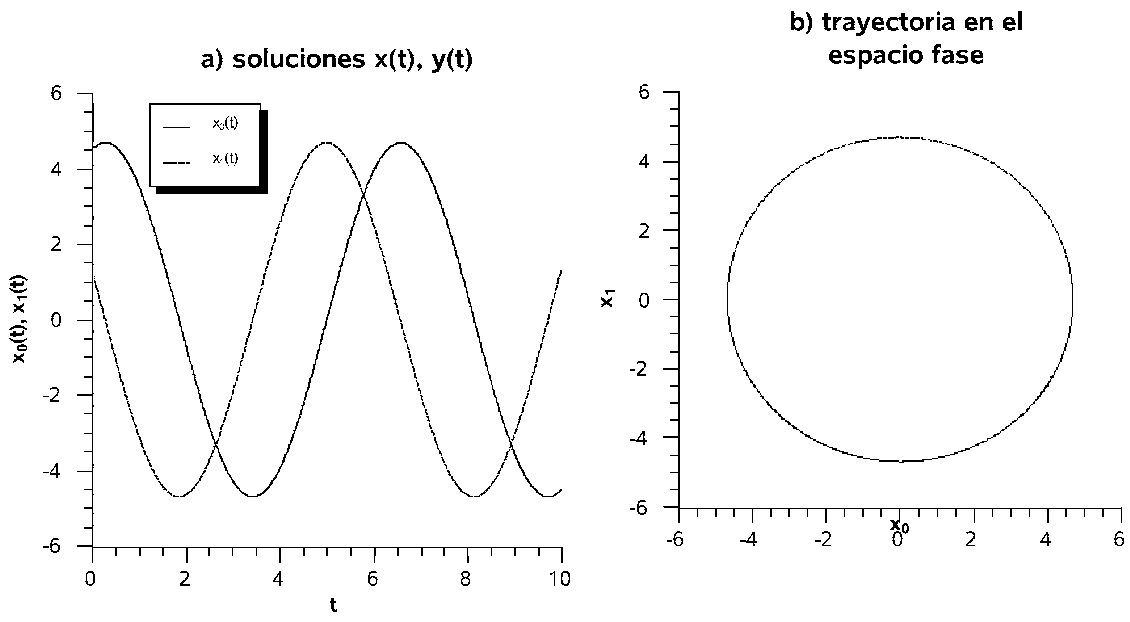

Figura 3. Soluciones a) tradicional y b) en el espacio fase sin fricción ($a = 0$).

Una cuestión interesante en términos de abstracción es la discusión de la dinámica del móvil a partir de la gráfica en el espacio fase, sobre todo comparando las soluciones para el caso en que el parámetro a es nulo con el caso amortiguado que se muestra en la figura 4.

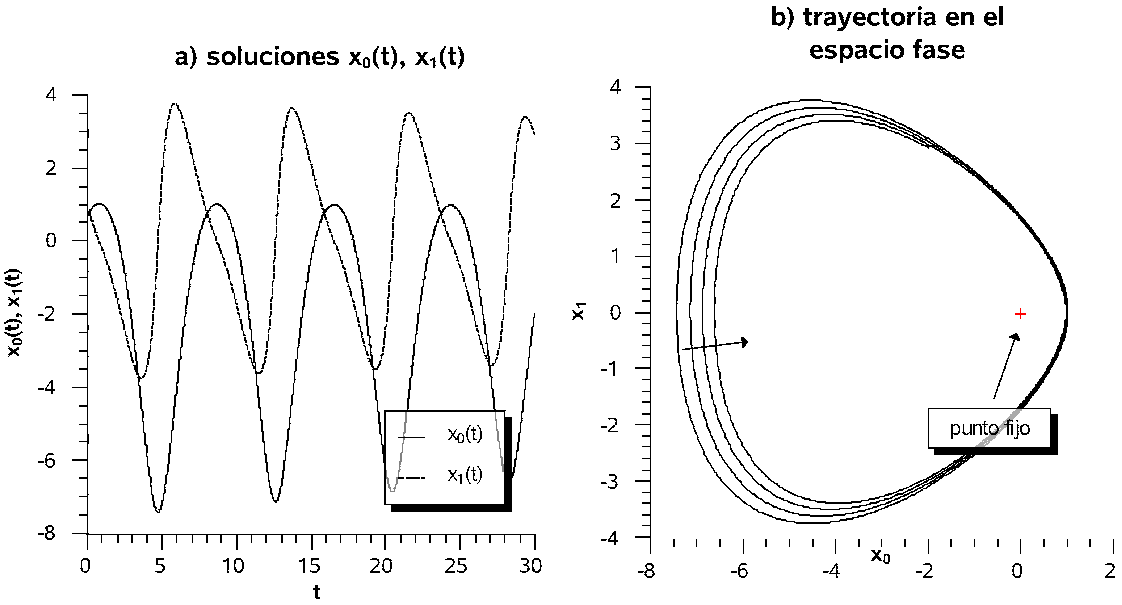

Figura 4. Las mismas soluciones para el caso $a = 1.6$, $b = 1.2$

Puede observarse que la trayectoria, además de deformarse, en cada ciclo se va reduciendo. El sistema se hará estable cuando alcance el punto fijo.

Usando `gnuplot` y la posibilidad de graficar un punto por vez de un archivo de datos, puede hacerse una animación en la que los estudiantes observan el movimiento del punto fase:

- en gnuplot, primer archivo «anima1.gp»:

```
set title 'espacio fase del oscilador armónico'
set xlabel 'x'
set ylabel 'v'
set xrange [-6:6]
set yrange [-6:6]
mit = 0
load 'anima2.gp'
```

- y en un segundo archivo «anima2.gp»:

```
plot 'datosF.dat' every ::mit::mit
mit = mit+1
if (mit < 5000) reread
```

Con una pequeña modificación este par de scripts permiten que el punto trace la trayectoria que recorre.

Quepa como un comentario final del ejemplo el hecho de que, si la velocidad del móvil es variable, seguramente será porque hay fuerzas actuando cuya suma no es cero. Este caso puede corresponder con un sistema en el que la partícula está sujeta a varias (o muchas) fuerzas individuales o bien pertenece a un conjunto en el que cada una de las partículas interactúa con todas las demás: cada una de ellas sufre los efectos de todas las demás en cada instante. Esta idea está más cercana a la Termodinámica Estadística Clásica, con la idea de *ensemble* o conjunto estadístico, etc. [12] y, en gneral, a los sistemas que no se consideran en equilibrio, compuestos de muchos elementos individuales o «partículas» que tienen interacción con las demás, que por la misma razón, en algún momento, presentan propiedades emergentes, que se caracterizan por comportamientos no lineales, etc. [6, 16]

4 Dinámica de muchas partículas

Una de las cuestiones que se pretende hacer notar es el hecho de que cada nuevo programa es una modificación del anterior. Las modificaciones pueden ser, desde incluir (o quitar) alguna(s) línea(s) al programa anterior, hasta cambios más profundos relacionados con el empleo de estructuras (listas), de funciones o de clases o, incluso, con la llamada a diferentes módulos. De la misma manera los conceptos están ligados, por ejemplo, por la cinemática (de una o de muchas partículas): puede verse que, exceptuando el caso del gas de esferas duras, todos los ejemplos están basados en la relación cinemática asociada con la regla de Euler (5) o la expreión (11).

4.1 Gas Ideal

Uno puede pensar de forma más o menos inmediata qué pasaría si tiene un montón de partículas con diferentes condiciones iniciales y que comienzan a moverse con velocidad constante: se irán dispersando hasta que finalmente todas se vayan. Más aún si no hay interacción entre ellas.

El gas ideal es un ejemplo en el que tenemos un conjunto enorme, en principio, de partículas que se mueven con velocidad constante y que no interactúan entre ellas. El primer ejemplo consiste entonces en un conjunto de partículas de masa unitaria en unidades arbitrarias. No hay interacción entre ellas ni hay campos externos. De este modo todas las partículas son «libres». Cada una de ellas se mueve con velocidad constante hasta que se encuentra con una pared dura, plana y perfectamente elástica de una caja bidimensional[11] de lado L con un vértice en el origen de coordenadas.

Hay que pensar entonces en una lista que tenga la información de las partículas: posiciones y velocidades al menos, aleatorias al inicio[12].

- en python (ver figura 3):

11. Obviamente esto es extendible a tres dimensiones. La idea aquí es poder visualizar el modelo usando el módulo Tkinter.

12. La distribución inicial podría ser la de equilibrio de Maxwell u otra a elección de quien construye la simulación. Aquí usaremos la distribución uniforme.

```
#!/usr/bin/python
# -*- coding: iso-latin-1 -*-
# Gas Ideal 2008
from Tkinter import *
from random import *

L=500 # lado de caja
np=20 # partículas
r=15  # radio
dt=0.01 # paso de tiempo

# gráfico
W=Tk()
C=Canvas(W, height=L, width=L, bg='black')
C.pack()
seed() # semilla de inicio random

# vectores de posición y velocidad
R=[[uniform(r,L-r),uniform(r,L-r)] for i in range(np)]
V=[[uniform(-L/2,L/2),uniform(-L/2,L/2)] for i in range(np)]

# las partículas serán círculos:
# (el simbolo '\' es para indicar que continua abajo)
p=[C.create_oval(R[i][0]-r, R[i][1]-r,R[i][0]+r, R[i][1]+r, \
   fill='red') for i in range(np)]

# se define la función de movimiento
def mover():
   for i in range(np):
      dx=V[i][0]*dt # cambios de posición
      dy=V[i][1]*dt
      C.move(p[i],dx,dy)
      # actualizar posiciones
      R[i][0]=R[i][0]+dx
      R[i][1]=R[i][1]+dy
      # verificar choque
      if R[i][0]>L-r or R[i][0]<r:
         V[i][0]=-V[i][0]
      if R[i][1]>L-r or R[i][1]<r:
         V[i][1]=-V[i][1]
   # se hace el ciclo... (modo gráfico)
   W.after(10,mover)
### fin mover()

# principal:
mover()
W.mainloop()
```

Este programa no hace cálculos, pero se puede evaluar con algunas modificaciones menores, por ejemplo la presión y la energía cinética promedio. Las dos primeras líneas del programa son para poder hacer el archivo ejecutable y para poder codificar caracteres acentuados, la «ñ», etc.

La parte gráfica (en general el uso de ventanas, botones, menú, canvas, etc.) se debe a una librería que normalmente se instala al instalar `python` y se llama Tkinter. En el listado anterior las líneas en que aparecen `W`, `C`, `p[]` están asociadas con objetos de la librería Tkinter [15, 10]

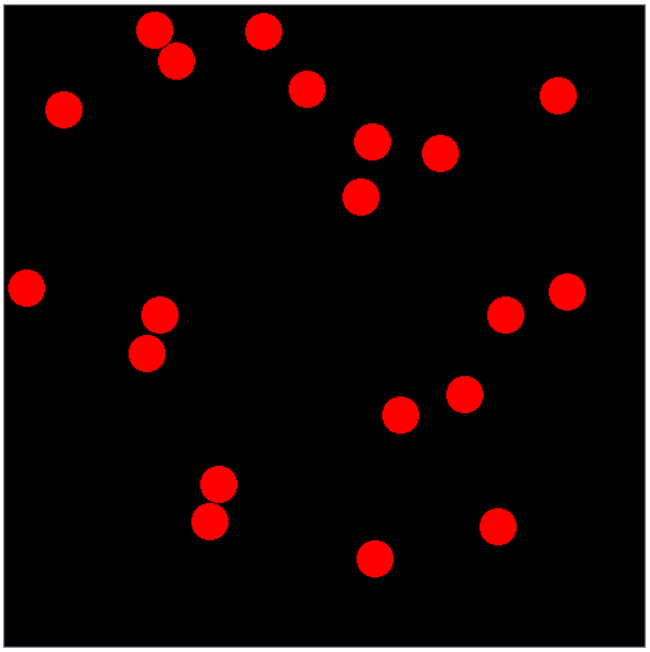

Figura 5. Imagen generada por el programa listado usando Tkinter

El efecto de las paredes, que podría representarse teóricamente con un potencial de pared dura (*hard core potential*) aquí se representa de forma simple considerando que las coordenadas de las partículas no excedan $[0, L] \times [0, L]$, en cuyo caso la componente de la velocidad normal a la pared cambia de signo. Para efecto visual el centro de masa no debe exceder el recuadro $[r, L - r] \times [r, L - r]$, de este modo las partículas no «cruzan» la pared. Esto implica que el volumen ocupable por las partículas será $(L - 2r)^2$. Una alternativa es hacer el lienzo (Canvas) de mayor dimensión. El hecho de poner volumen (radio) a la partículas ideales también es para efecto visual, no obstante hay que dejar claro que, cuando se trata modelos de gases o fluidos que pretenden ser más realistas, estas consideraciones, entre otras, se vuelven importantes.

La fuerza que actúa sobre una pared perpendicular al eje x (una línea de longitud L) debida al choque de una partícula está dada por

$$F_x \Delta t = \Delta p_x = 2 m v_x.$$

Si pensamos que Δt es el tiempo que tardan en viajar las partículas que hay desde la otra pared hasta llegar a ésta[13], entonces $\Delta t = L/v_x$ y entonces [7]

$$F_x L = 2 m v_x^2$$

o, de otro modo: la fuerza que actúa sobre una de las paredes será proporcional al cuadrado de la componente x de la velocidad. La fuerza total que actúa sobre la pared será entonces la fuerza promedio de todas las partículas por el número de partículas que han chocado en el intervalo Δt, es decir, la mitad de ellas. De este modo se tiene

$$\begin{aligned}\langle F_x\rangle &= 2m\frac{n}{2L}\langle v_x^2\rangle \\ &= m\frac{n}{L}\langle v_x^2\rangle.\end{aligned}$$

Esta suposición implica que los promedios de ciertas cantidades tienen sentido en términos de que el promedio coincide con el valor más probable (el que más aparece) y éste a su vez con el valor medido. Aunque en la naturaleza y en el mundo en general son dadas a aparecer distribuciones para las cuales el promedio no dice nada [13, 3]. Finalmente la presión:

$$P = \langle F_x\rangle/L = m\rho\langle v_x^2\rangle, \tag{12}$$

donde ρ es la densidad numérica de partículas. Así que bastará con sumar los cuadrados de todas las velocidades en alguna dirección y contar los choques para poder promediar.La fuerza que actúa sobre una pared perpendicular al eje x (una línea de longitud L) debida al choque de una partícula está dada por

$$F_x\Delta t = \Delta p_x = 2\mathrm{mv}_x.$$

Si pensamos que Δt es el tiempo que tardan en viajar las partículas que hay desde la otra pared hasta llegar a ésta[14], entonces $\Delta t = L/v_x$ y entonces [7]

$$F_x L = 2m\,v_x^2$$

o, de otro modo: la fuerza que actúa sobre una de las paredes será proporcional al cuadrado de la componente x de la velocidad. La fuerza total que actúa sobre la pared será entonces la fuerza promedio de todas las partículas por el número de partículas que han chocado en el intervalo Δt, es decir, la mitad de ellas[15]. De este modo se tiene

$$\begin{aligned}\langle F_x\rangle &= 2m\frac{n}{2L}\langle v_x^2\rangle \\ &= m\frac{n}{L}\langle v_x^2\rangle.\end{aligned}$$

Esta suposición implica que los promedios de ciertas cantidades tienen sentido en términos de que ese promedio coincide con el valor más probable (el que más aparece) y éste a su vez con el valor medido[16]. Finalmente la presión:

$$P = \langle F_x\rangle/L = m\rho\langle v_x^2\rangle, \tag{13}$$

13. Esto asegura que todas las partículas (salvo alguna que se mueva solamente en dirección y) han tocado a alguna de las paredes. O obien la mitad ha tocado una de las paredes. Esta idea permite asignar un valor a Δt.

14. Esto asegura que todas las partículas (salvo alguna que se mueva solamente en dirección y) han tocado a alguna de las paredes. O bien la mitad ha tocado una de las paredes. Esta idea permite asignar un valor a Δt.

15. De ser el caso tridimensional sería un tercio de las partículas.

donde ρ es la densidad numérica de partículas. Así que bastará con sumar los cuadrados de todas las velocidades en alguna dirección y contar los choques para poder promediar.

- esta versión la definición de una *clase*. Casi todo en el programa son declaraciones dentro de la clase `GasIdeal`. La acción se determina al final del archivo con la función `mainloop()`:

```
#!/usr/bin/python
# -*- coding: iso-latin-1 -*-
# Gas Ideal 2008
from Tkinter import *
from random import *
nc = 0
vmc = 0
class GasIdeal:
   def __init__(self, master=None):
      self.W=master self.L=500 # lado de caja
      self.np=20 # partículas
      self.r=15 # radio
      self.dt=0.01 # paso de tiempo
      seed()
      r, L=self.r, self.L
      self.R=[[uniform(r,L-r),uniform(r,L-r)] for i in \
              range(self.np)]
      self.V=[[uniform(-L/2,L/2),uniform(-L/2,L/2)] for i in \
              range(self.np)]
      # gráfico
      self.C=Canvas(self.W, height=self.L, width=self.L, \
                    bg='black')
      self.C.pack()
      self.b= Button(self.W, text='detener', command=self.salir)
      self.b.pack()
      # las partículas serán círculos:
      R=self.R self.p=[self.C.create_oval(R[i][0]-r,R[i][1]-r, \
        R[i][0]+r,R[i][1]+r,fill='red') for i in range(self.np)]
      self.mover()
   ### init

   def salir(self):
      global nc, vmc
      self.W.quit()
      print '====================='
      print 'No. de choques: ',nc
      rho = float(self.np)/float(self.L-2*self.r)**2
      vmc = vmc/float(self.np) print 'Presión: ',rho*vmc
      print '====================='
   ### salir

   # se define la función de movimiento
   def mover(self):
```

16. Es importante notar que en la naturaleza, en el mundo en general, son dadas a aparecer distribuciones para las cuales el promedio pierde este sentido [13, 3].

```
            global nc, vmc
            for i in range(self.np):
               dx=self.V[i][0]*self.dt
               dy=self.V[i][1]*self.dt
               self.C.move(self.p[i],dx,dy)
               # actualizar posiciones
               self.R[i][0]=self.R[i][0]+dx
               self.R[i][1]=self.R[i][1]+dy
               # verificar choque
               # en x se cuenta el choque y se suma Vx^2
               if self.R[i][0]<self.r or self.R[i][0]>self.L-self.r:
                  self.V[i][0]=-self.V[i][0]
                  nc = nc + 1
                  vmc = vmc + self.V[i][0]**2
               if self.R[i][1]<self.r or self.R[i][1]>self.L-self.r:
                  self.V[i][1]=-self.V[i][1]
            # se hace el ciclo... (modo gráfico)
            self.W.after(100,self.mover)
         ### mover
      ### class GasIdeal

      # principal:
      W=Tk()
      W.title('Gas Ideal...')
      miGasIdeal=GasIdeal(W)
      miGasIdeal.W.mainloop()
```

En el programa se suponen masas unitarias y se hace la corrección del volumen haciendo $\rho = n/(L - 2r)^2$. En la última parte se abre una ventana (Tk) y se declara un objeto (`miGasIdeal`) de la clase `GasIdeal`, enviando como parámetro el ámbito donde se llevará a cabo la acción (la ventana `W`).

4.2 Movimiento Browniano

Se puede hacer un modelo de Movimiento Browniano pensando que, en el recipiente del ejemplo anterior, se colocaran algunas partículas de mayor masa que las del gas, pero no tan grande que no puedan afectarles los choques de las partículas del fuido [7]. El efecto de las partícula pequeñas al chocar con las grandes será que éstas últimas se moverán de manera impredecible: más rápido o más lento y en cualquier dirección.

Una posibilidad computacional consiste en modificar la función `mover()` del programa del Gas Ideal para simular movimiento Browniano. No requiere velocidades. Hay que usar una función `random()` o `randint()` para decidir la dirección de movimiento de cada partícula (una de cuatro o bien de ocho posibles direcciones). Normalmente el tamaño de paso es constante, pero se puede hacer aleatorio. Si se supone que el movimiento Browniano es una superposición de partículas Brownianas independientes

- en python, la función `mover()`:

```
def mover():
   global t, Ct
```

```
      for i in range(np):
         gama=randint(0,3)
         dx = dy = 0
         if gama == 0: dx = 5
         elif gama==1: dx=-5
         elif gama==2: dy=5
         else: dy=-5
         C.move(p[i],dx,dy)
         # actualizar posiciones
         R[i][0]=R[i][0]+dx
         R[i][1]=R[i][1]+dy
      # correlacion <|R(t)-R(to)|>
      Ct=Ct+sum([ sqrt(R[j][0]**2+R[j][1]**2) \
                        for j in range(np) ])
      t=t+1
      print >> f, t, Ct
      # se hace el ciclo... (modo gráfico)
      W.after(10,mover)
   ### fin mover()
```

Aquí se ha supuesto que el tamaño de paso es `5`. Por simplicidad usamos la escala en pixels del Canvas como la escala de nuestro esperimento numérico de modo que, en cada «paso» una partícula se desplaza 5 pixels. En cada paso de tiempo se puede evaluar la correlación espacial[17] $C(t) = \langle |\boldsymbol{R}(t) - \boldsymbol{R}(t_0)| \rangle$ y verificar que, efectivamente, es proporcional al tiempo, en acuerdo con la solución de Einstein [5]:

Figura 6.

5 Uso de módulos de `python`

La implementación de aplicaciones se vuelve algo más sencillo de hacer si se tienen a mano módulos o librerías («bibliotecas») que contengan las funciones adecuadas. En esta sección discutimos el caso de un gas de esferas duras en el que se aplica el módulo `visual` (`vpython` que, a su vez, está construido sobre módulos como `numeric` o `numpy`[18] [REF]).

El gas de esferas duras (*Hard Spheres*) es un conjunto de partículas semejante al gas ideal, pero con una diferencia fundamental: ahora las partículas tienen volumen y este hecho las hace interatuar como bolas de billar, es decir, a través de choques entre ellas. Este hecho daría lugar a una ecuación de estado algo diferente a la del gas ideal, pues considera el volumen *excluído*, el volumen que se pueden ocupar $P(V - b) = NRT$

17. En el programa ejemplo se asume que las partículas parten del origen, de mdo que $\boldsymbol{R}(t_0) = (0, 0)$ para todas las partículas.

18. Estos últimos a su vez dependen de librerías tradicionales en fortran como BLAS y LAPACK.

Una cuestión sobresaliente es el uso de vectores que se pueden construir a partir de tuplas o de listas

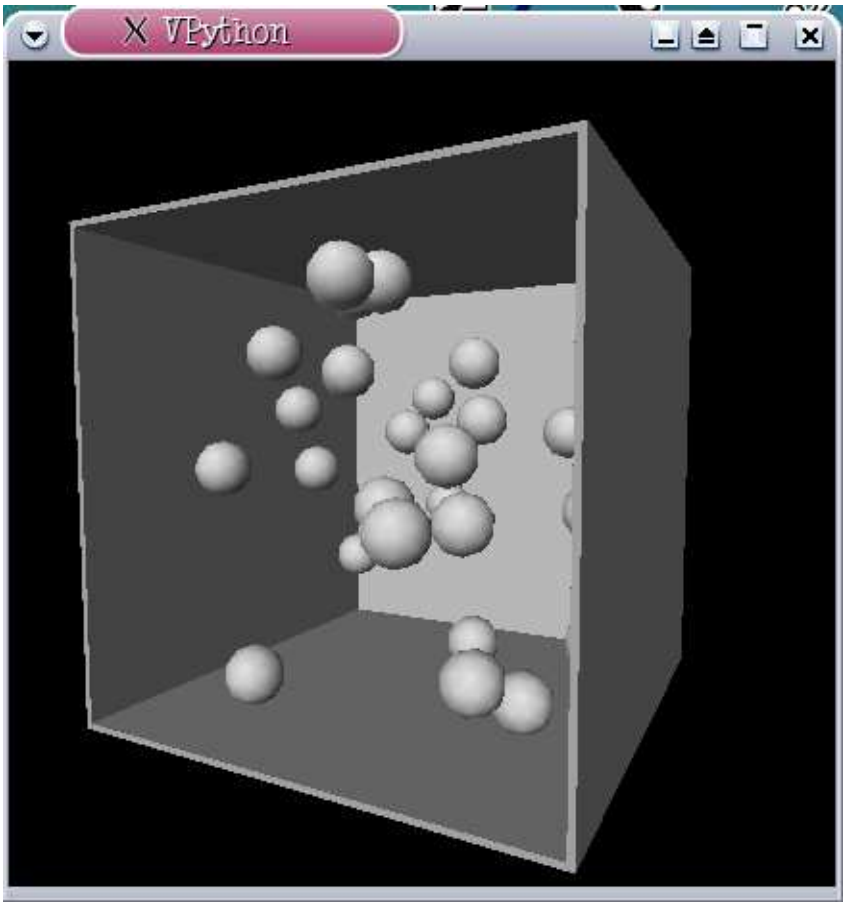

Figura 7. Gas de esferas duras en una caja. El gráfico está hecho con `visual python`.

6 Preguntas de control

Es claro que en este trabajo estamos resumiendo muchas experiencias y casos que se han dado (sobre todo en la Facultad de Ciencias Físico matemáticas de la BUAP, pero también en la Facultad de Ingeniería Química campus Acatzingo y Puebla) en unas cuantas líneas y ejemplos. Es así que en cada ejemplo, programa o sección hay muchos detalles que no se dicen, algunos deliberadamente ya en la experiencia docente, otros por falta de espacio o por criterio. Planteamos aquí una breve lista de preguntas específicas sobre los tópicos desarrollados en este artículo, sobre las omisiones o dudas que podrían generarse.

i.

ii.

iii. En el programa de la secicón 3.2 aparece una condición inicial (x_0, y_0) ¿Dónde aparece este punto en las gráficas de la solución y del espacio fase?

iv. ¿Por qué definimos el rango de los ejes en el intervalo [-5:5] en el caso del oscilador amortiguado al final de la sección 3.3[19]? o bien ¿Cómo podemos asegurar que la imagen del espacio fase está contenida en [-5, 5] × [-5, 5]?

v. Del problema anterior ¿Qué habría que cambiar en «anima2.gp» para que el punto fase fuese dibujando su trayectoria? (esto requiere de leer el manual de `gnuplot`)

vi.

vii. Si se definiera el *camino libre medio* como la máxima distancia que puede recorrer una partícula, en promedio, sin sufrir un choque. ¿Cómo se compara el camino libre medio en el gas ideal y en el gas de esferas duras (hacer una estimación para cada sistema)? ¿Cómo se podría calcular en ambos casos?

7 Comentarios y conclusiones

Bibliografía

[1] Página de python.

[2] Arnd and Báker. Computational physics educaction with python. *Computing in Science & Engineering*, pages 30–33, 2007.

[3] Nino Boccara. *Modeling Complex Systems*. Springer-Verlag, 2004.

[4] David M. Cook. Computation in undergraduate physics: The lawrence approach. *Am. J. Phys.*, 76(4&5):321–326, 2008.

[5] Albert Einstein. *Investigationns on the Theory of the Brownian Movement*. Dover Publications Inc., 2nd. edition, 1956.

[6] Gary William Flake. *The Computational Beauty of Nature (Computer Explorations of Fractals, Chaos, Complex Systems and Adaptation)*. The MIT Press, 2001.

[7] S. Frish and A. Timoreva. *Curso de Física General. Tomo I*. Editorial MIR, 3 edition, 1977.

[8] Rubin H. Landau. Resource letter cp-2: Computational physics. *Am. J. Phys.*, 76(4&5):296–306, 2008.

[9] Hans Petter Langtangen. *Python Scripting for Computational Science*. Springer-Verlag, 2004.

[10] Fredrik Lundh. *An Introduction to Tkinter*. Fredrik Lundh, December 1999.

[11] Jerry B. Marion. *Introduction to Classical Mechanics*.

[12] Donald D. Mc Quarrie. *Statistical Thermodynamics*.

[13] M. E. J. Newman. Power laws, Pareto distributions and Zipf's law. *arXiv:cond-mat/0412004*, 2004.

[14] Heinz-Otto Peitgen, Hartmut Jürgens, and Dietmar Saupe. *Chaos and Fractals (New Frontiers of Science)*. Springer-Verlag, 1992.

[15] John W. Shipman. *Tkinter reference: a GUI for Python*, December 2006.

19. Archivo «anima1.gp» del ejemplo de animación con `gnuplot`.

[16] Steven H. Strogatz. *Nonlinear Dynamics and Chaos*. Westview Press, 1994.

[17] Todd Timberlake and Javier E. Hasbun. Computation in classical mechanics. *Am. J. Phys.*, 76(4&5):334–339, 2008.