Mecánica 3d: `python` y el algoritmo de Verlet

J. F. Rojas y R. Martínez
Facultad de Ciencias Físico Matemáticas, Benemérita Universidad Autónoma de Puebla,
Av. San Claudio y 18 sur C.U. Col. San Manuel, 72570, Puebla, Pue. México.
e-mail: frojas@fcfm.buap.mx

M. A. Morales
Facultad de Ingeniería Química, Benemérita Universidad Autónoma de Puebla,
Av. San Claudio y 18 sur C.U. Col. San Manuel, 72570, Puebla, Pue. México.
e-mail: spinor70@yahoo.com.mx

Received 19 February 2013; accepted 5 May 2014

En el proceso de aprendizaje están involucrados varios hechos y actividades importantes. Algunas acciones como la discusión, la exposición de ideas y el desarrollo de proyectos, así como la posible solución de diferentes problemas planteados pueden ser complementadas con el uso de tecnologías comunes, hechos experimentales y materiales visuales didácticos. Consideramos que la discusión de diferentes proposiciones o técnicas para resolver una tarea específica contribuye a la interacción entre los estudiantes y, en el caso de los algoritmos de cómputo, los capacita para construir sus propias herramientas experimentales. Diferentes situaciones, preguntas, resultados o predicciones pueden ser propuestos y discutidos durante y después de la construcción de un programa de cómputo que presenta, en tiempo real, una visión animada tridimensional de la solución numérica. En este trabajo presentamos una deducción de los algoritmos de Verlet y los aplicamos a algunos sistemas mecánicos particulares, comenzando con la partícula libre y agregando, cada vez, algunas consideraciones físicas que incorporan, de manera gradual, complejidad al problema.

Descriptores: Modelación computacional; simulación; cómputo educacional.

In the process of learning there are several important events and activities involved. Some actions such as discussion, exposition of ideas and the development of projects or the possible solution of different shown problems can be complemented with the use of common technologies, experimental facts and visual didactic materials. We consider that the discussion of different propositions or techniques that solve some specific tasks contributes to the interaction between students and, in the case of computer algorithms, enable them to build their own experimental tools. Different situations, questions, results or predictions can be proposed and discussed during and after the construction of a computer program that shows in real time an animated visual 3D view of the numeric solution. We present deduction of Verlet's algorithms and apply them to some particular mechanical systems begining with free particle, and adding in each time some physical considerations that involves more complexity in the problem.

Keywords: Computational modelling; simulation; educational computing.

PACS: 45.10.-b; 05.45.Pq; 01.50.H-

1. Introducción

La mayoría de los problemas que enfrenta la ciencia en estos días se encuentran íntimamente vinculados con el uso y la aplicación de las computadoras. Ya sea para resolver parte de algún problema (evaluación de una integral, resolución de un sistema de ecuaciones, etc.) o bien, después de tener el modelo correspondiente, construir una solución del problema lo más completa posible. Otra posibilidad en el estudio de diferentes sistemas consiste en: partiendo de ciertas hipótesis y parámetros, realizar una simulación que nos permita observar comportamientos que probablemente no podríamos ver en un laboratorio o por medio de una solución analítica.

Tanto la solución de un modelo en la manera tradicional (las variables como función del tiempo) o en el espacio natural de la dinámica (*el espacio fase*[i]), como la simulación de eventos o fenómenos más allá de la materia inanimada son de gran importancia, en general, en el ámbito de la investigación científica.

La intención de los algoritmos de Verlet (o por lo menos de quienes los hemos usado) es la de resolver las ecuaciones de movimiento de sistemas formados por muchas partículas en los que pueden haber interacciones de diferentes formas (gravitacional, eléctrica, dipolar, de colisión, etc.) e incluso entre diferentes tipos de partículas. Por mencionar un caso, se puede pensar en un sistema con un solvente neutro (cuyos átomos no interactúan o lo hacen de manera muy débil) en el que se encuentran diluidos iones positivos y negativos en la misma cantidad. El problema se puede hacer más y más complejo, si, por ejemplo, se introduce una placa conductora sobre la cual los iones inducirán, en cada instante, distribuciones superficiales locales de carga que, obviamente, incidirán en la dinámica de los iones cercanos. Los tamaños y sobre todo la geometría de las partículas participantes son una complicación adicional muy difícil de tratar de manera analítica o incluso numérica [3,4].

El principio general de los algoritmos de Verlet consiste en resolver las ecuaciones de movimiento de cada una de las partículas que incluyen las interacciones, en principio, con todas las demás. La suma de todas las interacciones de to-

das las partículas, junto con los campos exteriores, sobre una partícula elegida resultará en la fuerza neta que actúa, por un instante (dado que todas las partículas se encuentran en movimiento continuo), sobre ésta. La técnica computacional que hace uso de esta idea se conoce como Dinámica Molecular. Esta técnica es aplicada principalmente en la simulación de sistemas de muchas partículas en los que se asume, o se propone, el tipo de interacción que interviene entre cada par de partículas[ii].

Por un lado, en este artículo pretendemos mostrar, con ejemplos típicos de la mecánica clásica, la aplicación y utilidad de las ecuaciones que conforman el algoritmo de velocidades de Verlet el cual, básicamente, nos dice cuánto cambia la posición de un cuerpo y cuánto su velocidad, en un intervalo pequeño de tiempo, cuando ese cuerpo se encuentra sujeto a fuerzas. Otro aspecto que rescatamos en este trabajo es la propuesta de los programas correspondientes bajo la idea de que puedan ser implementados en clases ordinarias o en sesiones tipo taller donde, a través de la cooperación y la discusión, se logren avances significativos en el entendimiento de los fenómenos de la mecánica y en la conceptualización teórica correspondiente. Las soluciones propuestas aquí no son, desde luego, únicas y están pensadas de manera que, cada algoritmo o programa, pueda modificarse de manera mínima para resolver otro problema más complejo. Se propone un conjunto de problemas o preguntas a manera de ejercicio colectivo o personal (tarea) además de las que surjan en la dinámica del curso.

La Sec. 2 está dedicada a la deducción de los algoritmos de *Leap-Frog* y *de velocidades* de Verlet, mientras que la Sec. 3 a desarrollar e implementar ejemplos de aplicación del último de estos algoritmos a diferentes casos de interés (campo gravitacional, colisiones elásticas e inelásticas, paredes, etc.). El problema de dos cuerpos está discutido en la Sec. 4; en la Sec. 5, se ejemplifica cómo plantear y resolver numéricamente el problema de tres cuerpos. La generalización de estas ideas puede dirigir de manera directa a la implementación de programas de Dinámica Molecular para sistemas de muchas partículas.

2. Las ecuaciones de movimiento discretas

Bajo ciertas circunstancias los problemas planteados no tienen una solución analítica. De manera particular, problemas asociados con ecuaciones no lineales tienen pocas posibilidades de ser resueltos con expresiones cerradas, en principio exactas. Dos ejemplos concretos y elementales dentro de la Mecánica son: el péndulo simple, que suele reducirse -para efectos teóricos- al caso de oscilaciones pequeñas, y las fuerzas de fricción, que suelen depender de potencias de la velocidad, lo cual vuelve al problema, como mencionamos antes, no lineal y sin posibilidades de ser resuelto de manera cerrada en la gran mayoría de los casos.

La propuesta de una solución aproximada a diferentes problemas resulta, entonces, una necesidad impostergable. La idea es que esta solución aproximada nos permita discernir sobre el comportamiento del sistema bajo estudio con la menor cantidad de error posible y, para muchos problemas reales, sin la necesidad explícita de un laboratorio. Requerimos de la solución, por un lado, que haga una buena descripción del evento o fenómeno y, por otro, la posibilidad de predecir[iii] el comportamiento futuro del sistema (al menos dentro de un ámbito restringido).

2.1. Las fórmulas de Verlet

Mencionamos en un artículo anterior [5] la forma y el significado de una expresión simple que nos sirve como fundamento y como algoritmo para describir el movimiento rectilíneo uniforme en un espacio sin fuerzas que tiene como base la discretización del tiempo. Dado dt una cantidad pequeña, constante, se tiene

$$dx = vdt$$

o, en dos y tres dimensiones,

$$d\vec{r} = \vec{v}dt,$$

donde $\vec{v}$ es un vector constante que representa la velocidad del móvil mientras que $d\vec{r} = \vec{r}(t+dt) - \vec{r}(t)$ es la diferencia de posiciones en el espacio entre el instante t y el instante $t + dt$. Entonces

$$\vec{r}(t+dt) = \vec{r}(t) + dt\vec{v}$$

nos permite encontrar la posición en el tiempo $t+dt$ en términos de la posición al tiempo t y de la velocidad del móvil $\vec{v}$. Esta forma de calcular las posiciones sucesivas de un objeto, en pequeños pasos discretos de tiempo dt, es una aplicación del método de Euler de primer orden [6].

Una mejor aproximación, útil además cuando la velocidad no es constante, resulta de considerar las expansiones en serie de Taylor de las cantidades $\vec{r}(t+dt)$ y $\vec{r}(t-dt)$:

$$\vec{r}(t+dt) = \vec{r}(t) + dt\frac{d}{dt}\vec{r}(t) + \frac{dt^2}{2}\frac{d^2}{dt^2}\vec{r}(t) + \ldots \quad (1)$$

$$\vec{r}(t-dt) = \vec{r}(t) - dt\frac{d}{dt}\vec{r}(t) + \frac{dt^2}{2}\frac{d^2}{dt^2}\vec{r}(t) - \ldots \quad (2)$$

Sumando ambas ecuaciones y truncando la serie obtenida a partir de los términos de cuarto orden[iv], tenemos:

$$\vec{r}(t+dt) = 2\vec{r}(t) - \vec{r}(t-dt) + \vec{a}(t)dt^2, \quad (3)$$

donde

$$\vec{a} \equiv \frac{d}{dt}\vec{v}(t) = \frac{d^2}{dt^2}\vec{r}(t).$$

Además, una buena aproximación para la velocidad (a segundo orden de precisión) es obtenida de la resta de (1) y (2)

$$\vec{v}(t) = [\vec{r}(t+dt) - \vec{r}(t-dt)]/2dt, \quad (4)$$

donde $\vec{v}(t) = \frac{d}{dt}\vec{r}(t)$. Esta expresión equivale, sustituyendo el tiempo $t \to t + dt$, a

$$\vec{v}(t+dt) = [\vec{r}(t+2dt) - \vec{r}(t)]/2dt. \quad (5)$$

Las expresiones (3) y (5) conforman el algoritmo básico de Verlet o *Leap-Frog* [3,4].

Sin embargo, cuando el movimiento involucra fuerzas, se requieren conocer simultáneamente las dos cantidades que definen el estado de movimiento de la partícula (su posición y velocidad). Es decir, si conocemos las posiciones $\vec{r}_i(t_0)$ y velocidades $\vec{v}_i(t_0)$ con $i = 1, 2, 3\ldots$ de las partículas involucradas en el sistema en el tiempo t_0, entonces deberíamos poder conocer los valores de las posiciones y velocidades después de un intervalo dt de tiempo: $\vec{r}_i(t_0 + dt)$ y $\vec{v}_i(t_0 + dt)$. Esto es complicado en el algoritmo original de Verlet (3) y (5) puesto que para conocer la posición y la velocidad en $t + dt$ se requiere conocer la posición en $t, t - dt$ y $t + 2dt$.

Con el objetivo de tener las posiciones y velocidades evaluadas en el mismo instante -lo que hace más claro el algoritmo- vamos a transformar un poco las relaciones (3) y (5). De ésta última, sumando y restando algunos términos por conveniencia, tendremos

$$\begin{aligned}\vec{v}(t+dt) &= \frac{1}{2dt}[\vec{r}(t+2dt) - \vec{r}(t) + \vec{r}(t+dt) \\ &\quad - \vec{r}(t+dt) + \vec{r}(t-dt) - \vec{r}(t-dt)] \\ &= \frac{\vec{r}(t+dt) - \vec{r}(t-dt)}{2dt} \\ &\quad + \frac{\vec{r}(t+2dt) - \vec{r}(t) - \vec{r}(t+dt) + \vec{r}(t-dt)}{2dt},\end{aligned}$$

donde el primer término corresponde a $\vec{v}(t)$ según la expresión (4). Por otro lado, reemplazando el término $-\vec{r}(t+dt)$ por $-2\vec{r}(t+dt)+\vec{r}(t+dt)$ y el término $-\vec{r}(t)$ por $-2\vec{r}(t) + \vec{r}(t)$ se tiene

$$\begin{aligned}\vec{v}(t+dt) = \vec{v}(t) + \frac{dt}{2}\Bigg[&\frac{\vec{r}(t+2dt) - 2\vec{r}(t+dt) + \vec{r}(t)}{dt^2} \\ &+ \frac{\vec{r}(t+dt) - 2\vec{r}(t) + \vec{r}(t-dt)}{dt^2}\Bigg]\end{aligned}$$

que es lo mismo, de acuerdo a (3), que

$$\vec{v}(t+dt) = \vec{v}(t) + \frac{dt}{2}[\vec{a}(t+dt) + \vec{a}(t)]$$

y así, finalmente, las ecuaciones del *algoritmo de velocidades de Verlet* son:

$$\vec{r}(t+dt) = \vec{r}(t) + \vec{v}(t)dt + \vec{a}(t)dt^2/2 \tag{6}$$

y

$$\vec{v}(t+dt) = \vec{v}(t) + [\vec{a}(t) + \vec{a}(t+dt)]dt/2. \tag{7}$$

Una cuestión importante para fines prácticos es el siguiente hecho: lo que se puede evaluar o conocer *a priori* en el sistema son las fuerzas. Esto implica que las expresiones anteriores sean reescritas en términos de la fuerza neta que está actuando sobre la masa correspondiente (que se asume constante). De ahí que

$$\vec{r}(t+dt) = \vec{r}(t) + \vec{v}(t)dt + \vec{F}(t)dt^2/2m \tag{8}$$

y

$$\vec{v}(t+dt) = \vec{v}(t) + [\vec{F}(t) + \vec{F}(t+dt)]dt/2m, \tag{9}$$

nos permiten obtener resultados evaluando las fuerzas que intervienen en el problema. Hay que notar que las expresiones (8) y (9) pueden interpretarse como los incrementos (o los cambios) en la posición y en la velocidad de la partícula: si pasamos restando al lado izquierdo el primer término de cada una de las expresiones del lado derecho se tienen los cambios de posición y velocidad entre t y $t + dt$:

$$d\vec{r} = \vec{v}(t)dt + \vec{F}(t)dt^2/2m \tag{10}$$

y

$$d\vec{v} = [\vec{F}(t) + \vec{F}(t+dt)]dt/2m. \tag{11}$$

De esta manera, las expresiones (10) y (11) representan los incrementos correspondientes al desplazamiento y al cambio de velocidad que sufre una partícula bajo el efecto de una fuerza neta $\vec{F}$. En el caso de dos partículas con interacción mutua habrá que evaluar la fuerza una sola vez ya que, de acuerdo a la tercera ley de Newton, esta misma fuerza con sentido contrario, actuará sobre la otra partícula.

3. Aplicaciones elementales didácticas

En la presente sección se discuten e implementan los programas correspondientes a la dinámica de una sola partícula partiendo del caso más simple e incrementando la complejidad del problema en cada caso. Los ejemplos van de la partícula libre hasta una constreñida al movimiento dentro de una caja y sujeta a un campo gravitacional uniforme.

3.1. Partícula libre

El caso más simple que se estudia en la mecánica es el de la partícula libre: una partícula puntual (de preferencia) que, con cierta masa definida, se desplaza en el espacio manteniendo, por alguna razón[v], su velocidad $\vec{v}$ constante. En este caso las ecuaciones que «gobiernan» el movimiento de nuestra partícula se reducen significativamente: las fuerzas netas que actúan sobre la partícula son cero, así que (8) y (9) se convierten en:

$$\vec{r}(t+dt) = \vec{r}(t) + \vec{v}(t)dt \tag{12}$$

y

$$\vec{v}(t+dt) = \vec{v}(t)$$

o bien, si empleamos (10) y (11),

$$d\vec{r} = \vec{v}(t)dt$$

y

$$d\vec{v} = 0,$$

que equivale a $\vec{v} = \text{constante}$ lo cual define una trayectoria rectilínea con rapidez constante. Este problema, y el código correspondiente, ya fueron tratados en la Ref. 5.

3.2. Tiro parabólico

Un sistema mecánico simple consiste en una partícula que es lanzada con cierta velocidad inicial $\vec{v}_0$ que, como es obvio, se puede expresar en términos de sus componentes o bien de la rapidez y un ángulo de disparo. Una vez que la partícula es disparada solamente tiene interacción con el campo gravitacional terrestre (o cualquier otro, desde luego).

En este caso, se tiene una aceleración constante si se piensa en una partícula moviéndose cerca de la superficie terrestre y las ecuaciones de Verlet (10) y (11) quedan como:

$$d\vec{r} = \vec{v}(t)dt + \vec{g}dt^2/2,$$
$$d\vec{v} = [\vec{g} + \vec{g}]dt/2,$$

donde $m\vec{g}$ es una fuerza que no depende del tiempo y que, vectorialmente, puede representarse como $(0, -mg, 0)$, donde g es el valor numérico del campo gravitacional. Entonces, las ecuaciones pueden reducirse a:

$$d\vec{r} = \vec{v}(t)dt + \vec{g}dt^2/2,$$
$$d\vec{v} = \vec{g}dt, \qquad (13)$$

de modo que el algoritmo de una partícula sujeta al campo gravitacional, con condiciones iniciales dadas es[vi]:

LISTADO 1:

```
01  #_*_ coding: utf8 _*_
02   from visual import *
03   from math import *
04   # DECLARAIÒN DE VARIABLES FISICAS
05  t = 0.0
06  dt = 0.001
07  atirop = 2. * (pi/6.)
08  vo = 12
09  vx = vo* cos(atirop)
10  vy = vo*sin(atirop)
11  g = 9.8
12   m = 20.0
13  radio = 0.02
14  nops = 20   # A PARTIR
        DE AQUI VARIABLES DE ENTORNO
15  L = 15.
16  lc = L
17   Lb = (2.*lc)/3.
18  ## LONGITUD EJES DE REFERENCIA
19  win1 =750
20  ## ANCHO DE LA VENTANA
21  win2 = 550
22  angulo =.8 ## RANGO
        VISUALIZACION CAMARA
23  scene = display(title="Tiro parabolico",
            width=win1, \
            height=win2, x=1000,y=0, \
            range=(angulo*lc, angulo*lc, angulo*lc), \
            center=(5,3,0), background=(0,0,0))
24  axisX = arrow(pos=(0,0,0), axis=(Lb,0,0), \
            shaftwidth=0.001, color=color.red)
25  axisY = arrow(pos=(0,0,0), axis=(0,Lb,0),\
            shaftwidth=0.001, color=color.blue)
26  axisZ = arrow(pos=(0,0,0), axis=(0,0,Lb),\
            shaftwidth=0.001, color=color.green)
27  label(pos=(Lb,0,0), text='x')
28  label(pos=(0,Lb,0), text='y')
29  label(pos=(0,0,Lb), text='z')
30   vel=vector(vx,vy)
31   bola=arrow(radius=radio,pos=(0.,0.,0.),\
            axis=vel,color=color.white)
32  bola.vel = vel
33  peso = vector(0.,-m * g,0.)
34  # ciclo de la dinámica
35  while bola.pos.y ¿0.0:
36     rate(nops)
37     rant = vector(bola.pos)
            POSICION "ANTERIOR"
38      t = t + dt
39     dr=bola.vel*dt+peso*dt**2/(2.*m)
40     dv = fuerza*(dt/m)
41     bola.pos = bola.pos + dr
42     bola.vel = bola.vel + dv
42     bola.axis = bola.vel
43     trayectoria = curve(pos = [rant,bola.pos],
            color = \
            color.yellow, radius=0.05)
```

En este caso, como puede observarse en la Fig. 1, el «proyectil» (que no se ve) tiene «adherida» una flecha que está definida, todo el tiempo, por el vector velocidad. Hay que remarcar que las cantidades **`bola.pos`**, **`bola.vel`** y **`peso`** son vectores, de la misma manera que **`dr`** y **`dv`** que corresponden a las Ecs. (10) y (11) o bien, más específicamente, a (13) y (14) (ver líneas 39 a 42 del código en LISTADO 1). La fuerza es un vector constante (definido en la línea **`33`**) y, en este caso, se trata de una partícula puntual que nombramos **`bola`**. La imagen asociada al objeto es una flecha (**`arrow`** en la línea **`31`**) cuyo tamaño y dirección (**`axis`**) están determinadas por el vector velocidad **`bola.vel`**.

3.3. Fuerzas de fricción

Un caso más realista es el de un proyectil que, además de la interacción con el campo gravitacional, interactúa con el aire circundante. En este caso hay que considerar que sobre

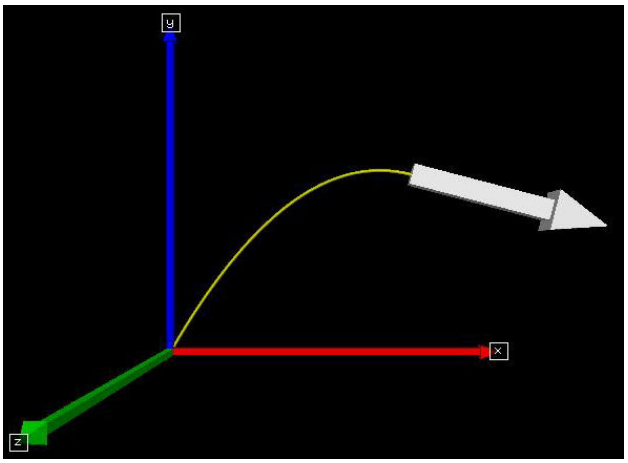

FIGURA 1. El vector velocidad se muestra todo el tiempo: define la rapidez y la dirección instantáneas del movimiento.

el proyectil actúa una fuerza de fricción de la forma genérica

$$\vec{f}_r = -kv^{\alpha}\hat{v},$$

donde $\hat{v}$ es el vector unitario en dirección de la velocidad y $v = |\vec{v}|$ la rapidez. Las constantes k y α dependen de las propiedades del medio (viscosidad, densidad, forma, etc.) y de la forma del objeto. Para este caso, hay que hacer unas modificaciones al programa anterior. En primer lugar, y dado que tenemos dos fuerzas que actúan sobre el proyectil, definiremos una función **`sumafuerzas()`** -que evalúa la fuerza neta sobre la partícula en cada instante de tiempo- en lugar del vector de fuerza constante de la línea **33** que dice: **`peso = vector(0.0,-m*g,0.0)`**. Además de esto se deben definir los valores de **`k`** y **`alpha`** (ver código más abajo). Entonces, la función **`sumafuerzas()`** debe evaluar todas las fuerzas que actúan sobre la partícula (esto puede ponerse inmediatamente abajo de la línea **33**):

```
# evaluacion de fuerzas sobre la particula
k = 0.1
alfa=1.
def sumafuerzas():
    global peso
    vUnitario = norm(bola.vel)
    fFriccion = - k * mag(bola.vel)**alfa * vUnitario
    s = peso + fFriccion
    return s
```

Hay que poner, para este caso, mucha atención a las ecuaciones que definen el algoritmo de velocidades de Verlet (8) y (9). De manera particular, la expresión (9) que evalúa la «nueva» velocidad $\vec{v}(t+dt)$ queda en términos de la fuerza evaluada en el instante t y en el instante $t+dt$. Las fuerzas en general *no* dependen del tiempo (por lo menos no de manera explícita), para este ejemplo la fuerza tiene una parte constante (el peso) y una parte que depende de la velocidad, es decir que $\vec{F}(t) = \vec{F}(\vec{v}(t))$. Obviamente, $\vec{F}(t+dt) = \vec{F}(\vec{v}(t+dt))$, así que requerimos evaluar la velocidad en dos instantes para evaluar la fuerza de fricción en esos dos instantes[vii]:

```
01  fuerza = sumafuerzas()
02   # CICLO DE LA DINÁMICA
03  while bola.pos.y > 0.0:
04      rate(nops)
05      rant = vector(bola.pos)
            # guarda la posicion "anterior"
06     bola.pos = bola.pos + bola.vel
            * dt + fuerza * dt**2/(2.*m)
07      bola.vel = bola.vel + fuerza
            * dt/m # la fuerza "anterior"
08   fuerza = sumafuerzas()
    # evalua la nueva fuerza
09  bola.vel = bola.vel + fuerza * dt/m
10  curve(pos = [rant,bola.pos],
    color = color.yellow, \
            radius = 0.05)
11   t = t + dt
```

En la línea **`01`** se evalúa la suma de fuerzas sobre la partícula para comenzar a «moverla», así que la línea **`06`** no requiere discusión (hasta este momento la fuerza está evaluada en t). La línea **`07`** evalúa el primer término de la nueva velocidad con la misma fuerza. Esta nueva velocidad (en $t+dt$) implica que la fuerza de fricción se modifique, así que evaluamos la fuerza en $t+dt$ en la línea **`08`** para, después, en la línea **`09`**, evaluar la segunda parte de (9). El resultado será una trayectoria consistente en una parábola como la de la Fig. 1, pero deformada a causa de la fricción.

Vamos a agregar dos características físicas más al proyecto: el proyectil será una pelota que rebota en el piso y, además, pierde energía en forma de calor en cada rebote. Ahora se tiene una partícula que interactúa con el medio (fricción), que colisiona con una superficie y que pierde energía como resultado de cada colisión. Esto implica que se detendrá cuando su energía se haga cero (ver Fig. 3).

En primer lugar, vamos a introducir una nueva función **`reboteP(p)`** que evalúe los efectos del «rebote» del proyectil con el suelo: cambio repentino de momento vertical y pérdida de energía cinética por calor. Hay que considerar, para construir la función, que estos cambios ocurren en el preciso momento en el que la coordenada y de nuestro proyectil se hace negativa si suponemos que $y = 0$ corresponde al plano del piso.

Antes de construir la función hay que definir algunos parámetros:

```
H=6.0     ## ALTURA INICIAL
alfa=1.   ## EXPONENETE DE LA VELOCIDAD DE
              LA FUERZA DE FRICCIÓN
```

```
k=0.1       ## COEFICIENTE DE FRICCIÓN
                CINEMÁTICO
beta=0.8    ## COEFICIENTE DE RESTITUCIÓN
```

lo cual implica, por ejemplo, que la línea **31** del **LISTADO 1** quede como

```
bola=sphere(radius=radio,
pos=vector(0.,H,0.),color=color.cyan)
```

y luego la función:

```
def reboteP(p):
global radio, beta
if p.pos.y < radio:
  p.pos.y = 2*radio - p.pos.y # corrige posición
  p.vel.y = -p.vel.y # cambio de momento en y
  p.vel = sqrt(beta)*p.vel # perdida de energia
return p.pos, p.vel
```

Si en cada colisión o rebote con el piso la partícula pierde un 20 %, por ejemplo, de su energía cinética, entonces el coeficiente de restitución será de **0.8**, lo cual significa que «recupera» el 80 % de su energía inicial. Si β es el coeficiente de restitución, entonces $E' = \beta E$ y, en consecuencia, $v'^2 = \beta v^2$. Por lo tanto, la velocidad después de cada colisión[viii] será $v' = \sqrt{\beta} v$. El efecto puede ser notado comparando la Fig. 3 con la Fig. 2. Si el lector desea una revisión más completa sobre colisiones entre partículas puede revisar la Ref. 7.

El ciclo principal ahora tiene que incluir el llamado de la función que evalúa las fuerzas y el llamado a la función que corrige la dinámica por posible colisión[ix] (ver Figs. 2 y 3).

```
01  fuerza = sumafuerzas()
02  # CICLO DE LA DINÁMICA
03  while True:
04      rate(nops)
05      rant = vector(bola.pos)
06      bola.pos = bola.pos + bola.vel*dt + 0.5
        *fuerza*dt**2./m
07      bola.pos, bola.vel = reboteP(bola) # corrige
        por colision
08      bola.vel = bola.vel + fuerzas * dt/m # verlet
        1er termino
09      fuerza = sumafuerzas()
        # nueva fuerza (t+dt)
10      bola.vel = bola.vel + fuerza * dt/m # verlet
        2o termino
11      curve(pos = [rant,bola.pos],color = color.white,
        radius=0.05)
12      t = t + dt
```

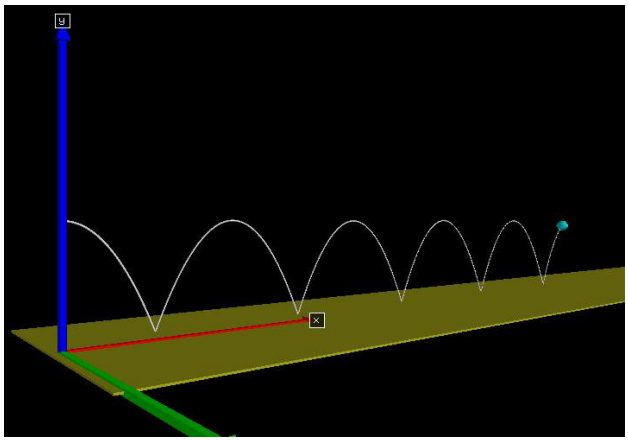

FIGURA 2. La pelota rebotando sin fricción y con coeficiente de restitución unitario.

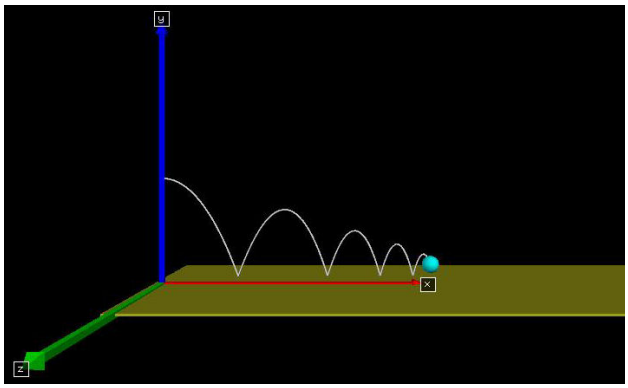

FIGURA 3. Efectos de la fricción con el aire y de la pérdida de energía cinética en cada colisión.

3.4. Paredes y campo gravitacional

Se aborda aquí un caso diferente al anterior: si no tenemos otra forma de contener a la partícula que metiéndola en una caja (lo cual es muy común en los sistemas de muchas partículas, comenzando por el gas ideal [8]), entonces habrá que construir un algoritmo que refleje (y muestre) el efecto de las paredes sobre la partícula[x].

Una de las cuestiones importantes en la construcción del algoritmo es la siguiente: se debe reflejar la física que se encuentra detrás de éste, es decir, lo que se quiere modelar o simular. En este caso comenzaremos por lo más simple (o lo más ideal) que corresponde a paredes adiabáticas, duras y lisas donde los choques de las partículas son completamente elásticos y especulares. En este caso, una partícula con posición y velocidad iniciales dadas se mueve dentro de una caja que constriñe su movimiento a permanecer dentro de ella. El siguiente paso será que la misma partícula se mueva bajo la acción del campo gravitacional local, considerado uniforme.

El caso de las paredes lisas, adiabáticas, etc. se resuelve algorítmicamente haciendo que la partícula que «cruce» una pared haga simplemente una reflexión especular sobre ella. Por ejemplo, si una partícula tiene velocidad[xi] $\vec{v} = (0, 0, 1)$

y está tan cerca de la pared que se encuentra paralela al plano XY (digamos en $z = 6$) entonces, al «tocarla», deberá cambiar su velocidad v_z por $-v_z$. Es decir, en términos del algoritmo cuando la coordenada z de la partícula esté «muy cerca» de 6 entonces hay que cambiar v_z por $-v_z$ de modo que la velocidad tras el impacto sea $\vec{v} = (0, 0, -1)$. Esto mismo hay que hacerlo para cada una de las caras de la caja.

Una cuestión importante aquí es el hecho de que las interacciones elásticas con las paredes no se consideran como fuerzas dentro del algoritmo: no hay una forma más o menos simple de expresar analíticamente este tipo de interacción. Un caso diferente correspondería a la interacción con una fuerza constante como es el caso del campo gravitacional $\vec{g}$:

LISTADO 2:

```
01  from visual import *
02  # lados de la caja -L,L en las tres direcciones
03  L = 20
04  # las paredes de la caja (se dibujan 5)
05  esp = 0.1 # espesor
06  s2 = 2*L - esp
07  # ancho de paredes
08  s3 = 2*L + esp
09  pR = box (pos=vector( L, 0, 0),
          length=esp, height=s2, \
          width=s3, color = (0.7,0.7,0.1))
10  pL = box (pos=vector(-L, 0, 0),
          length=esp, height=s2,\
          width=s3, color = (0.7,0.7,0.1))
11  pB = box (pos=vector(0, -L, 0),
          length=s3, height=esp, \
          width=s3, color = (0.7,0.7,0.1))
12  pT = box (pos=vector(0, L, 0),
          length=s3, height=esp, \
          width=s3, color = (0.7,0.7,0.1))
13  pBK = box(pos=vector(0, 0, -L),
          length=s2, height=s2, \
          width=esp, color = (0.8,0.8,0.2))
14  g = vector(0,-9.8,0)
15  t, dt, R = 0.0, 0.001, 0.2
16  p = sphere(radius=R,
          pos=(-10,5,0), color=color.white)
17  # agregamos atributo al objeto p
          (sphere): la velocidad
18  p.vel = vector(12,7,-1)
19  pos = p.pos # posicion auxiliar
20  # definimos LL para efecto visual
          (no traslape con pared)
21  LL = L - 0.5*esp - R # una
          caja menor para el centro d masa
22  def sgn(x):
23          if x > = 0.0 return 1.
24          else return -1
25  def impacto(p):
26          global pos, LL
27          s = sgn(pos.x)
28          if abs(pos.x) > LL:
29                  p.vel.x = -p.vel.x
30                  posx = s*abs(2*LL) - pos.x
31          s = sgn(p.pos.y)
32          if abs(pos.y) > LL:
33                  p.vel.y = -p.vel.y # nueva velocidad
34                  pos.y = sabs(2*LL) - pos.y
35          s = sgn(p.pos.z)
36          if abs(pos.z) > LL:
37                  p.vel.z = -p.vel.z # nueva velocidad
38                  pos.z = s*abs(2*LL) - pos.z
39  while t < 5000:
40          dr = p.vel*dt + 0.5*g*dt**2
          # suponiendo masa = 1
41          dv = g*dt
42          pos = p.pos + dr
          # nueva posición en auxiliar
43          impacto(p) # corrige posicion/velocidad
44          p.pos = pos # actualiza posicion particula
45          p.vel = p.vel + dv
46          t += dt
```

En este programa las líneas **`40, 41, 42`** y **`45`** corresponden explícitamente al algoritmo de Verlet de velocidades, pero en las líneas **`42-44`** se usa un vector auxiliar **`pos`** para evitar que, visualmente, las esferas se traslapen con las paredes: primero se evalúa la nueva posición en la lista auxiliar `pos`, se corrige con la función **`impacto(p)`** y luego se asigna al objeto **`p.pos = pos`** el valor corregido. Las líneas **`25-38`** modifican las velocidades en las direcciones[xii] x, y, z, del mismo modo que las posiciones. Basta con «desaparecer» la línea **`43`** y el efecto de las paredes en el programa desaparecerá (ver Fig. 4).

Hay que remarcar que el campo gravitacional sí aparece explícitamente dentro de las ecuaciones de Verlet, no así las paredes. El caso es que no hay, como ya dijimos, una forma analítica simple, con representación algorítmica, de representar una fuerza infinita.

3.5. El gas ideal: partículas libres constreñidas

El gas ideal es un sistema en el que no aparecen fuerzas explícitas sobre las moléculas que lo forman: en este caso sola-

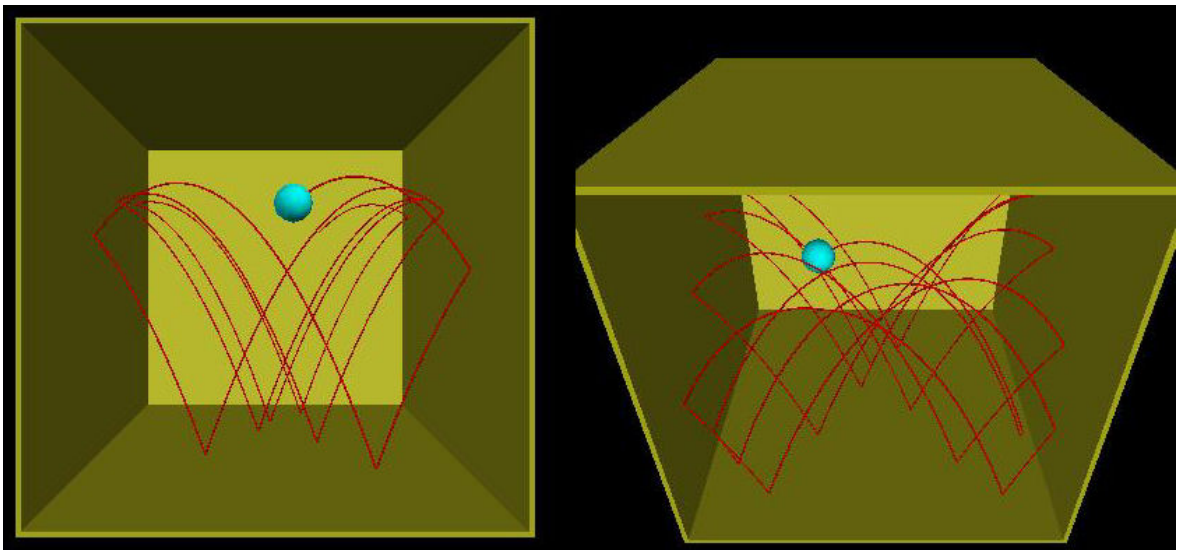

FIGURA 4. Dos momentos de la dinámica de una partícula en una caja con paredes elásticas y con campo gravitacional.

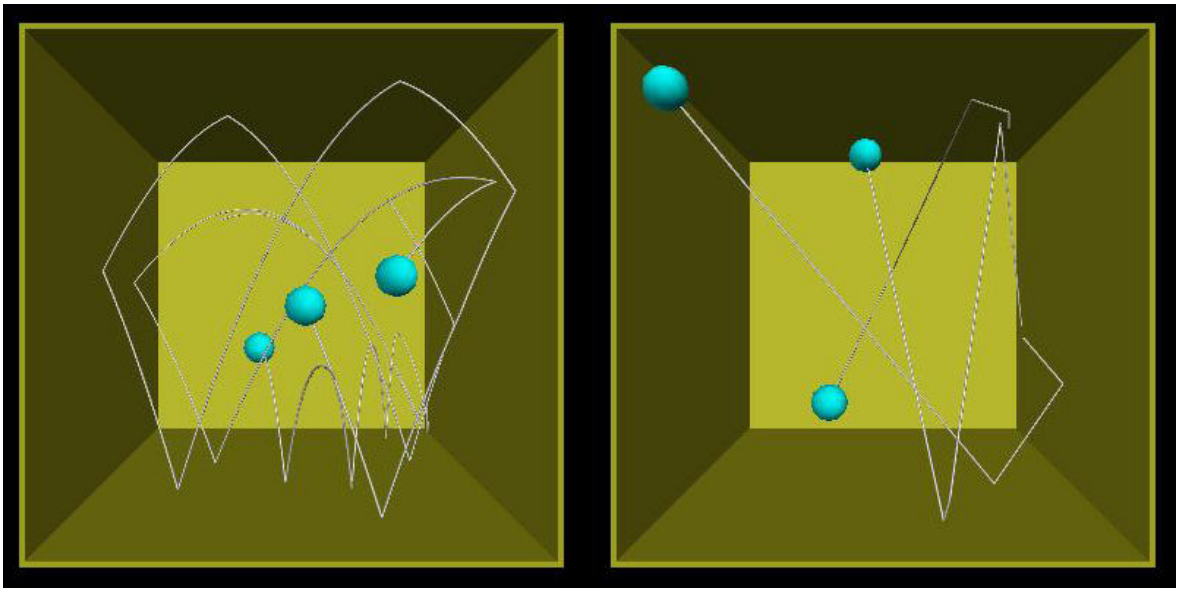

FIGURA 5. Partículas en una caja con y sin campo gravitacional. Nótese la diferencia de trayectorias.

mente existen las colisiones con las paredes completamente elásticas y lisas que son las encargadas de modificar el momento lineal de cada una de las partículas.

El efecto de cambio de momento lineal (o de la velocidad, puesto que la masa de las partículas no cambia) en la dirección perpendicular a la pared de impacto ya fue discutido en la sección anterior. En este caso vamos a discutir brevemente la construcción de una lista de partículas, para lo cual basta con modificar unas cuantas líneas de código del **`LISTADO 2`**: de las primeras líneas de código puede verse que la caja se encuentra entre $-L$ y L para las tres coordenadas, así que para colocar aleatoriamente a las partículas dentro de la caja basta con definir **`np = 50`** si, por ejemplo, se quieren cincuenta partículas. Conviene definir entonces una función que nos devuelva un número aleatorio entre $-LL$ y LL[xiii].

```
def u():
    global LL
    return uniform(-LL,LL)
```

antes de la línea **`14`**. La «frase» **`global LL`** sirve para que la función **`u()`** tome el valor de **`LL`** que es una variable ***externa*** a la función. El objeto de definir esta función es escribir menos código en la línea **`16`**: ésta será sustituida por una lista de **`np`** objetos **`sphere`**:

```
p = [ sphere(radius=0.2, pos=( u(),u(),u() ),
    color=color.white) \ for s in range(np) ]
```

que son idénticos a `p` con la salvedad de que sus posiciones son aleatorias en el intervalo $[-LL, LL] \times [-LL, LL]$

$\times [-LL, LL]$, que es el espacio dentro de la caja que puede ser ocupado por el centro de masa de cada partícula. Las líneas **20-27** definen la dinámica de una sola partícula bajo un campo gravitacional $(0, -g, 0)$ y con paredes elásticas. Esta misma dinámica funcionará para las **np** partículas si, recorriendo estas líneas a la derecha, insertamos una línea (la **19**) que recorra a todos los elementos de la lista **p**:

```
18  while t < 5000:
19          for s in p: # “para
            todo objeto s en p”
20                  dr = s.vel*dt + 0.5*g*dt**2
21                  dv = g * dt
22                  pos = s.pos + dr # nueva posición
23                  impacto(s)
24                  s.pos = pos
29                  s.vel = s.vel + dv
30          t += dt
```

En este ejemplo, el radio de las partículas solamente se considera en la interacción con las paredes y únicamente como un efecto visual pues, se supone, estamos tratando con partículas o masas puntuales. Lo que se representa en este modelo es un gas de partículas que no tienen interacción entre sí (no “se ven” y, por lo tanto, en la animación, se pueden traslapar). Al hacer **g = vector(0,0,0)** se tiene el típico gas ideal, como puede observarse en la Fig. 5.

4. El problema de dos cuerpos

El problema de dos cuerpos que interactúan por su campo gravitatorio generalmente se reduce (cuando los tamaños de las masas no son comparables) al de un solo cuerpo sujeto al campo de interacción del otro (de mucho mayor masa). Esto oscurece conceptos como el de Centro de Masa (CM) de varias partículas y puede llevar a confusión en problemas más avanzados de la dinámica de muchas partículas. Un ejemplo común es el caso en el que los estudiantes no visualizan el hecho de que, sin fuerzas externas, el CM no debe moverse. En esta sección se discute el problema de dos partículas involucrando en el código la dinámica del centro de masa (ver Fig. 7).

4.1. Dos partículas bajo interacción mutua

Si uno pregunta a un grupo de estudiantes que reflexionen sobre cómo harían la descripción de dos cuerpos o partículas aisladas que tienen interacción mutua (gravitacional por ejemplo) las respuestas pueden variar entre describir el movimiento de una fijando al observador en la otra o bien elegir un punto con respecto al cual describir ambas. La primera respuesta tiene el inconveniente de que la descripción no se hace desde un sistema de referencia inercial.

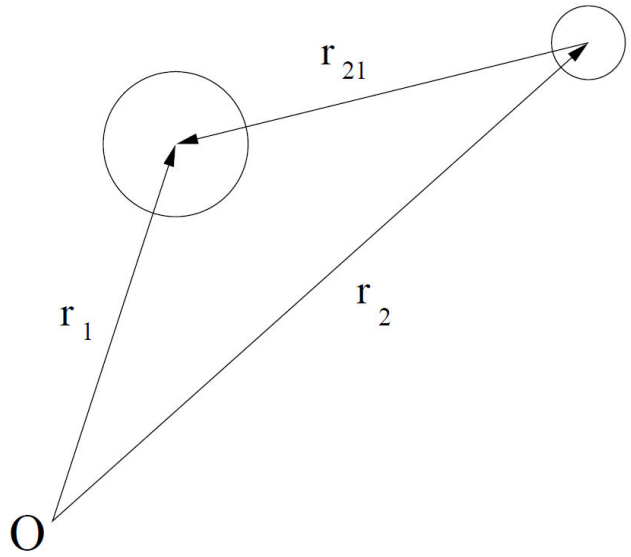

FIGURA 6. Dos masas diferentes interactuando bajo una fuerza de atracción constante dirigida a lo largo del vector $\vec{r}_{21} = \vec{r}_1 - \vec{r}_2$.

En un curso de Mecánica los estudiantes pueden llegar a algunas conclusiones, a partir de la discusión de un ejemplo numérico visual, con respecto al papel de las fuerzas internas en un sistema de partículas interactuantes. La discusión teórica respecto al papel del centro de masa e, incluso, de la justificación de lo que se llama *partícula puntual* cobran mucho sentido después de esta experiencia. El problema concreto se plantea en términos de modificar los programas previos para describir lo que pasa con, precisamente, dos partículas de diferente masa y tamaño que sufren una interacción de atracción mutua, de intensidad constante y dirigida a lo largo de la línea que une sus centros de masa:

La fuerza que ejerce la partícula 2 sobre la 1 es $\vec{F}_{21} = -|\vec{F}|\hat{r}_{21}$ donde $\hat{r}_{21} = \frac{\vec{r}_{21}}{|\vec{r}_{21}|}$ es el vector unitario que une a la partícula 2 con la 1 y, de acuerdo con la tercera ley de Newton, $\vec{F}_{12} = -|\vec{F}|\hat{r}_{21}$ es la fuerza que ejerce la partícula 1 sobre la partícula 2. En la construcción del algoritmo se debe considerar, entonces, la magnitud (invariante) de la fuerza de atracción mientras que el vector $\vec{r}_{21}$ se evalúa en cada instante de tiempo dado que $\vec{r}_{21}(t) = \vec{r}_1(t) - \vec{r}_2(t)$ (ver Fig. 6).

El programa es el siguiente:

```
1  #!/usr/bin/env python
2  #-*- coding: iso-latin-1 -*-
3  from visual import *
4  " " "
5  Problema: resolver el problema
           de dos cuerpos que ejercen una
6  fuerza mutua atractiva constante
           dirigida a lo largo de la linea
7  que los une:
8  la solucion se logra resolviendo
           con el algoritmo de Verlet
9  " " "
```

```
10  # masas de las particulas
11  m1,m2 = 10.0,10.0
12  # magnitud de la fuerza mutua constante
13  fmag = 123.8
14  # vector de campo gravitacional
15  g = vector(0.,-9.8,0.)*0
16  # condiciones iniciales
17 r1=vector(10.,0.,0.) r2=vector(-5.,0.,0.)
18  # velocidades iniciales
19 v1=vector(-1.,3.,0.) v2=vector(1.,3.,0.)
20  # paso de tiempo
21 dt=0.0001
22 # objetos graficos
23 display(title='dos cuerpos con
          atracción constante', width=600, ´
          height=600, center=(0,0,0), background=(1,1,1))
24  # ejes
25 curve(pos=[(-10,0,0),(10,0,0)],radius=0.2)
26 curve(pos=[(0,-10,0),(0,10,0)],radius=0.2)
27  curve(pos=[(0,0,0),(0,0,10)],radius=0.2)
28 # particulas
29 s1=sphere(pos=r1,color=(0.9,0.5,0.5),radius=m1/5.)
30  s2=sphere(pos=r2,color=(0,0.9,0),radius=m2/5.)
31 # centro de masa
32 Rcm = (m1*r1 + m2*r2)/(m1+m2)
33 rcm = sphere(radius=0.3,pos=Rcm,
          color=color.black)
34 raw_input() # espera <enter>
35  # dinamica
36 while 1:
37    # vector unitario de r2 a r1
38     r21u = norm(r2-r1)
39     # las fuerzas "anteriores"
40     f21 = fmag*r21u + m1*g
41     f12 = -fmag * r21u + m2*g
42     # algoritmo de verlet
       para las dos particulas
43     r1 = r1 + v1*dt + 0.5*f21/m1*dt**2
44     r2 = r2 + v2*dt + 0.5*f12/m2*dt**2
45     # vector unitario de
       r2 a r1 en las nuevas posiciones
46     r21u = norm(r2-r1)
47     # las nuevas fuerzas
48     f21_n = fmag * r21u + m1*g
49     f12_n = -fmag * r21u + m2*g
50     # hay colision en la
         posicion nueva de las dos particulas?
51     if mag(r2-r1) < s1.radius + s2.radius:
52       v1x = -(m1-m2)*v1/(m1+m2) + 2*m2*v2/(m1+m2)
53       v2x = 2*m1*v1/(m1+m2) + (m2-m1)*v2/(m1+m2)
54       v1, v2 = v1x, v2x
55   # las velocidades
         incluyendo las fuerzas en t y t+dt
56   v1 = v1 + 0.5*dt* (f21+f21_n)/m1
57   v2 = v2 + 0.5*dt* (f12+f12_n)/m2
58   # centro de masa
59   Rcm = (m1*r1 + m2*r2)/(m1+m2)
# trayectorias
 curve(pos=[s1.pos,r1],
 radius=0.1,color=(0.9,0.5,0.5))
curve(pos=[s2.pos,r2],
radius=0.1,color=(0.0,0.9,0.0))
curve(pos=[rcm.pos,Rcm],
radius=0.1,color=(0.0,0.0,0.0))
 # actualizacion de objetos graficos
 rcm.pos = Rcm s1.pos=r1 s2.pos=r2
 " " "
Solucion de una colision frontal:
vf1 = (m1-m2) v1 /(m1+m2) + 2 m2 v2/(m1+m2)
vf2 = 2 m1 v1 / (m1+m2) + (m2-m1) v2/(m1+m2)
 " " "
```

El resultado presenta dos aspectos: el primero es el hecho de que aunque las dos partículas no siguen trayectorias «descriptibles» de manera simple, el centro de masa sí lo hace: el centro de masa se mueve en una típica trayectoria del movimiento rectilíneo uniforme[xiv]. El otro aspecto es el poco realismo que se observa cuando las dos partículas se acercan, debido a la atracción constante, y se traslapan espacialmente. Aunque este caso no se muestra aquí puede ser un buen pretexto para discutir colisiones pues los objetos reales no se traslapan. En el caso de colisiones elásticas la solución en una dimensión requiere un par de ecuaciones algebraicas: la que corresponde a la conservación del Momento Lineal y la de la conservación de la Energía. La solución en una dimensión de una colisión frontal[xv] en la que la partícula i tiene masa m_i y velocidad antes de la colisión v_i ($i = 1, 2$) es:

$$v_1' = \frac{m_1 - m_2}{m_1 + m_2} v_1 + \frac{2m_2}{m_1 + m_2} v_2,$$

$$v_2' = \frac{2m_1}{m_1 + m_2} v_1 + \frac{m_2 - m_1}{m_1 + m_2} v_2,$$

donde las v_j' representan las velocidades inmediatamente después de la colisión. La expresión es extensible a tres dimensiones como [9]

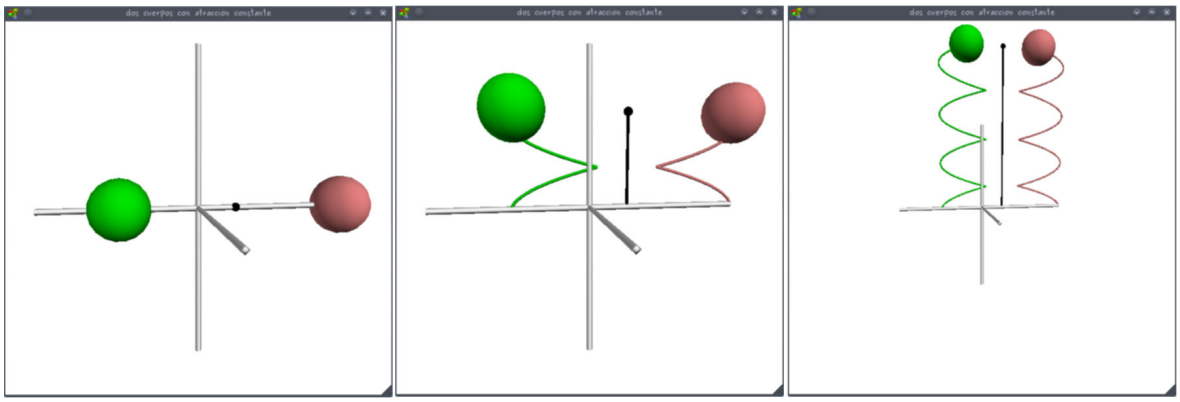

FIGURA 7. Solución del problema de dos partículas con una fuerza atractiva constante. La primera imagen muestra la condición inicial. La línea negra creciendo verticalmente entre las esferas es la trayectoria del Centro de Masa.

$$\vec{v}_1' = \frac{m_1 - m_2}{m_1 + m_2}\vec{v}_1 + \frac{2m_2}{m_1 + m_2}\vec{v}_2, \tag{14}$$

$$\vec{v}_2' = \frac{2m_1}{m_1 + m_2}\vec{v}_1 + \frac{m_2 - m_1}{m_1 + m_2}\vec{v}_2. \tag{15}$$

Esta condición puede ser agregada al algoritmo y al programa de dos partículas logrando un efecto, en general, más realista y por lo mismo convincente de que los algoritmos, dentro de la precisión y contexto definidos, son correctos[xvi] (revisar las líneas **50-54** del código y ver Fig. 7).

4.2. Interacción gravitacional

La interacción gravitacional entre dos masas puntuales i y j está dada por la Ley de Gravitación Universal de Newton [2]

$$\vec{F}_{ij} = -\frac{Gm_i m_j}{r_{ij}^2}\hat{r}_{ij}, \tag{16}$$

donde $\hat{r}_{ij}$ es el vector unitario en la dirección de $\vec{r}_j - \vec{r}_i$, mientras que r_{ij} es la distancia entre las partículas i y j, es decir, $r_{ij} = |\vec{r}_j - \vec{r}_i|$. Introducir esta interacción en el modelo anterior lo cambia de escala, literalmente, pues ya se puede hablar de cuerpos astronómicos y no de dos partículas sujetas a un campo mutuo de fuerza constante.

En la siguiente sección presentamos, por último, una versión más compleja en cuanto a construcción algorítmica, pero más simple en términos de programación. Se trata de un programa en el que se hace uso de la librería `numpy` de `python` para simular el comportamiento, en el espacio real, de tres partículas diferentes cuya interacción mutua es puramente gravitacional. Realizando varios experimentos puede observarse que, efectivamente, cambios pequeños en las condiciones iniciales pueden llevar al sistema a comportamientos radicalmente diferentes, es decir, el sistema presenta caos[xvii].

5. El problema de los tres cuerpos

A finales del siglo XlX Henri Poincaré [2], tratando de aplicar la ley de la atracción universal a tres cuerpos (que podrían ser el Sol, la Tierra y la Luna), tal como ya se había hecho para dos cuerpos, se encontró con un problema sumamente complejo cuyo resultado variaba ostensiblemente con sólo pequeñas variaciones de las distancias entre los cuerpos. Así, Poincaré encontró las características que hoy llamamos caóticas del problema de los tres cuerpos, introduciendo lo que el llamó soluciones doblemente asintóticas, de manera análoga a las encontradas por E. Lorenz alrededor de 1960 en modelos relacionados con el clima [10,11]. Al parecer, las técnicas que utilizó reflejan en mucho los métodos actuales de la Teoría del Caos.

Genéricamente, el problema puede plantearse considerando tres partículas aisladas en el espacio que tienen interacción mutua, gravitacional por ejemplo. De acuerdo con la Fig. 8 los vectores $\vec{r}_{21} = \vec{r}_1 - \vec{r}_2$, $\vec{r}_{31} = \vec{r}_1 - \vec{r}_3$ y $\vec{r}_{32} = \vec{r}_2 - \vec{r}_3$ definen la dirección instantánea (considerando que las tres partículas se mueven) de la fuerza gravitacional entre las correspondientes partículas, de acuerdo con la expresión (17).

Aunque ya discutimos el caso de partículas esféricas con volumen en términos de esferas duras[xviii] la realidad es que

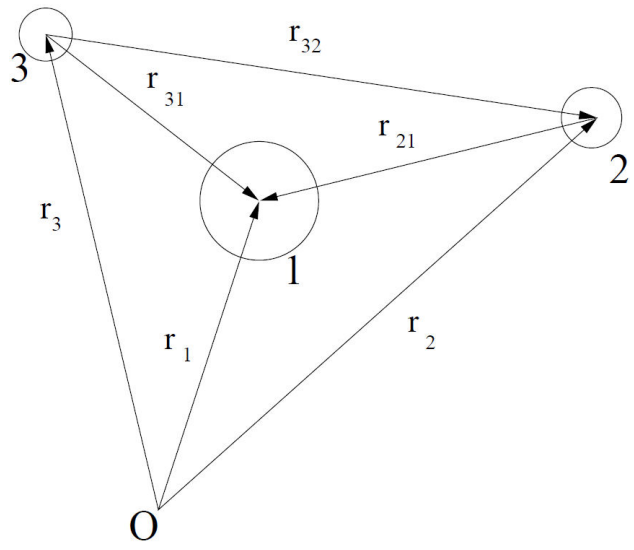

FIGURA 8. Esquema para tres partículas y sus interacciones.

todos los materiales son deformables en alguna cuantía. Esto implica que si la energía del impacto entre dos partículas es grande la deformación lo será también. Este hecho se modela con un potencial genérico que representa la «penetrabilidad» o deformación de las partículas que se hace mayor a mayores energías de impacto, superpuesto a la interacción gravitacional de Newton.

Pensemos primero que colocamos una de las partículas en el origen de coordenadas, de manera que r es un eje o coordenada que parte del centro de la partícula en cualquier dirección^{xix}. Cualquier otra partícula sufrirá el efecto del campo gravitacional de la primera si se coloca a una distancia dada r de la misma. Es decir, se considera la energía potencial gravitacional como cero cuando $r \to \infty$ y a la energía potencial gravitacional, que de acuerdo con

$$\Delta U(r) = -\int_{r_{\text{ref}.}}^{r} \vec{F}_g(r') \cdot d\overrightarrow{r'}$$

es

$$U(r) = -\frac{GMm}{r}. \tag{17}$$

Se le agrega un término que haga las veces del núcleo esférico de la primera partícula que corresponda a una barrera «impenetrable» de energía potencial o, equivalentemente, a una fuerza repulsiva enorme cuando los centros de masa de ambas partículas se aproximan «demasiado». Para el efecto se propone un potencial de la forma

$$U(r) = -GMm\left[\left(\frac{s}{r}\right)^n - \frac{s}{r}\right] \tag{18}$$

donde $s = R_i + R_j$ es la distancia de máximo acercamiento entre dos de las partículas de radios R_i y R_j mientras que n es un parámetro que puede cambiarse en el código. El segundo término de la expresión (19) corresponde al potencial gravitacional.

El código `python` correspondiente es el siguiente (ver Figs. 9 y 10):

```
#-*- coding: utf8 -*-
from visual import *
from random import uniform as u

s = 1.
### parametros del potencial
g = 30.
n = 2
no_particles = 3
maxV = 5.0
D3 = 1 # define el movimiento en 3 dimensiones

Lmin=1.1
L=25 # L largo en x y ancho en z de la celda
Ly = 2.0*L # Ly alto de la celda
maxpos=6
Lx=Ly
rc=L/2
Raxes = 0.2
dt =0.01
zaxis2 = curve(pos=[(Lx,Ly,0),
      (Lx,-Ly,0), (-Lx,-Ly,0), (-Lx,Ly,0),
      (Lx,Ly,0)], color=color.white, radius=Raxes)
#####################################################
# declaración de listas de posicion,
      velocidades, masas y radios para las esferas
balls=[]
vlist=[]
poslist=[]
M = array([12,5,8])
R = 0.3*M
m_1 = array([1./M[i] for i in
      range(no_particles)]) # div por m en
                            # Verlet
c=[color.yellow,color.red,color.blue]
for i in arange(no_particles):
    ball = sphere(color = c[i], radius=R[i]) # los objetos
    p=[maxV*u(-1,1), maxV*u(-1,1),
      D3*maxV*u(-1,1)] # velocidades
    vlist.append(p) # posiciones:
    position=[maxpos*u(-1,1), maxpos*u(-1,1),
      D3*maxpos*u(-1,1)]
    poslist.append(position)
    ball.pos=vector(position)
    balls.append(ball)
# conversion a array
velarray=array(vlist) # velarray=velocidades-iniciales
posarray=array(poslist) # posarray=posiciones-iniciales
##############################
# declaracion de funciones
### potencial
def potencial(i, j):
    global g,n,R
    s = R[i]+R[j]
    rij = posarray[i]-posarray[j]
    r = mag(rij)
      return g*M[i]*M[j]*((s/r)**n-(s/r))
### la fuerza por pares...
def Fuerza(i, j):
      global g,n,R
      s = R[i]+R[j]
```

```
        rij = posarray[i]-posarray[j]
        r = mag(rij)
        return g/r*M[i]*M[j]*(n*(s/r)**n-(s/r))*rij/r
### Energia
def energia():
        global posarray, velarray
        np = no_particles # para escribir menos
        # calculo de las energias potencial y cinetica
        U = sum([potencial(i,j) for i
        in range(np) for j in range(np)])
        K = sum([0.5*M[i]*mag(velarray[i])**2
        for i in range(np)])
        while K > abs(U)/2:
            velarray *= 0.5
            K = sum([0.5*M[i]*mag(velarray[i])**2
               for i in range(np)])

###################################
# modulo para el calculo de fuerzas
def cfuerzas():
        f=[0]*no_particles
        for i in range(no_particles-1):
            for j in range(i+1,no_particles):
            Fij = Fuerza(i,j)
            f[i] = f[i] + Fij
            f[j] = f[j] - Fij
        farray=array(f)    #cálculo de fuerzas iniciales
        return farray

farray=cfuerzas()    # las fuerzas
        iniciales sobre las particulas
h=0
energia()     # energia del sistema al inicio
while h <10000:     # "while 1:"para un loop 'infinito'
        rate(100)
        posant = posarray.copy()
        posarray += velarray*dt
        + 0.5*farray*m_1*dt**2 # posicion Verlet
        velarray += 0.5*farray*m_1*dt
        # velocidad Verlet a t farray = cfuerzas()
        velarray += 0.5*farray*m_1*dt

 # velocidad Verlet a t+dt
 for i in range(no_particles):
     balls[i].pos = posarray[i]
     curve(pos=[posant[i],posarray[i]], color=c[i], radius=0.5)
 h += 1
```

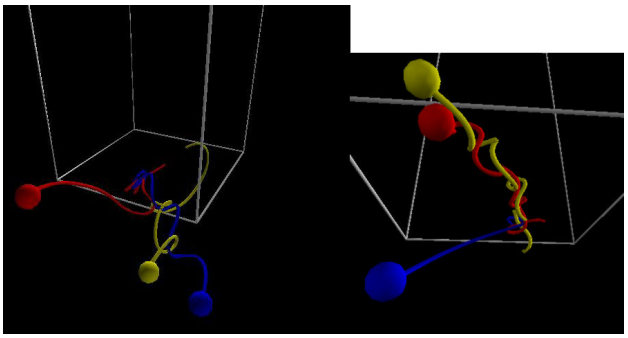

FIGURA 9. Trayectorias de tres partículas con condiciones iniciales diferentes: se trata de un sistema caótico (altamente sensible a las condiciones iniciales).

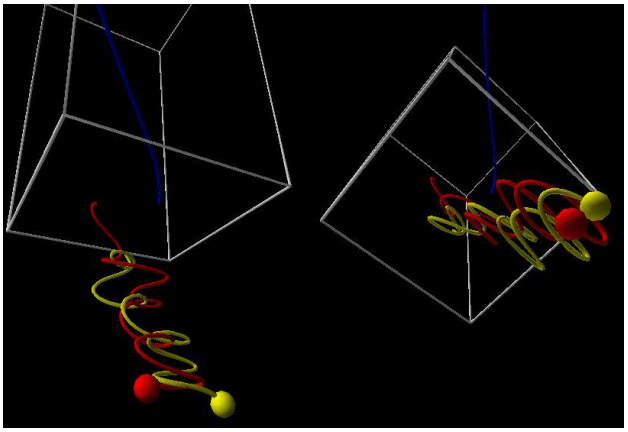

FIGURA 10. Bajo ciertas condiciones una de las partículas escapa; las otras dos orbitan a la manera de una estrella doble.

En el código se ha incluido una función **`energia()`**. Aunque no representa efectos sobre el sistema de ejemplo, con este artificio se puede lograr cierto control sobre el sistema en términos del predominio de las fuerzas atractivas, o bien de la dispersión que se lograría por agitación térmica. Esto podría usarse para modelar o simular gases o líquidos, efectos de condensación, etc. ya sea proponiendo diferentes formas del potencial de interacción (o las fuerzas) u observando la posibilidad alternativa de disminuir el valor de la rapidez máxima **`maxV`**.

6. Preguntas de control

A diferencia del enfoque del artículo [5], en este caso aparecen involucradas de manera explícita las fuerzas que toman parte en la dinámica de cada ejemplo. Este hecho permite que los estudiantes investiguen diferentes situaciones y comportamientos dinámicos haciéndose preguntas y modificando los parámetros del programa.

En esta sección proponemos una lista de cuestiones ya sea a manera de preguntas, propuestas o ejercicios que son más bien una guía de todas las cosas que podrían discutirse en clase, aun antes de experimentarlas en los programas:

- ¿Por qué en el tiro parabólico (**`LISTADO 1`**) el tamaño de flecha cambia? ¿Dónde es mínimo? ¿Por qué?
- Verificar, donde sea posible, lo que debería ocurrir si **`|g| = 0.0`** (no hay campo gravitacional).
- Discutir con detalle las ideas que están involucradas en la función **`reboteP(p)`**: la geometría es importante para esta y otras simulaciones.
- Verificar qué ocurre si se quita una pared en todos los casos posibles.
- ¿Cómo hacer un hoyo en una de las paredes y permitir que las partículas, en su caso, salgan por ahí? ¿Podría evaluarse la densidad dentro de la caja como función del tiempo?
- Un problema muy interesante: ¿Qué condiciones se requieren para que el sistema de tres cuerpos se comporte aproximadamente como un sistema Sol-Tierra-Luna?

7. Comentarios

Los programas de ejemplo que proponemos, desde luego, no son la única ni la mejor manera de resolver los modelos. Sin embargo, creemos que pueden servir como base para, primero, discutir y descubrir formas de implementar ideas y conceptos físicos en un programa de simulación, que tiene la gran ventaja de la imagen animada y tridimensional. Además, creemos que lo más importante es, aunado a la práctica computacional tan importante para el quehacer científico hoy, la discusión en el aula que, finalmente, es la que va a enriquecer el conocimiento de los estudiantes ya sea clarificando o aportando ideas nuevas a cada uno, de todos para todos.

Estas prácticas -muchas aún bajo diseño- han logrado que estudiantes que se encuentran alrededor de la mitad de la licenciatura se animen a realizar proyectos propios que han servido, con un poco de esfuerzo adicional, para presentar trabajos en los congresos nacionales de la SMF[xx] o para becas de programas locales como el de «Jóvenes Investigadores» y «La ciencia en tus manos» que promueve la Vicerrectoría de Investigación y Estudios de Posgrado de la BUAP[xxi].

---

i. De acuerdo con la formulación con que se trabaja, el *espacio fase* puede ser el espacio constituido por todas las coordenadas y las velocidades (formulación Lagrangiana), o bien por todas las coordenadas involucradas en el sistema y los momentos asociados con ellas (formulación Hamiltoniana). Ver, por ejemplo [1,2].

ii. En una simulación las fuerzas o interacciones que intervienen deben ser evaluadas a pares: la fuerza que actúa sobre la partícula i, $\vec{F}_i$, es la resultante de todas las fuerzas $\vec{F}_{ji}$ de cada partícula j sobre la indicada i, es decir $\vec{F}_i = \sum_{j\neq i} \vec{F}_{ji}$. Esta situación es obligada por el hecho de que no existe una solución analítica al problema de tres cuerpos interactuando simultáneamente [2].

iii. En general, no solamente se busca una descripción o entendimiento de cualquier evento o fenómeno. Generalmente la actividad se encamina a la búsqueda de la predicción -o incluso del control- del sistema.

iv. Nótese que al sumar ambas expresiones los términos de orden uno y tres en dt son eliminados, quedando solamente los de orden par. El cortar de la serie los términos de orden cuatro y mayor hace que el error en la aproximación sea del orden de $\mathcal{O}(dt^2)$.

v. Físicamente esto implica que, o bien la partícula está libre de interacciones (no hay cuerpos o campos) o que la suma vectorial de todas las posibles interacciones (cuerpos, campos, etc.) que pueda sufrir la partícula es cero: las fuerzas se anulan.

vi. Todos los programas mostrados están escritos para `python` versión 2.x. Las versiones 3.x corresponden a una versión alterna del mismo lenguaje.

vii. El algoritmo de Verlet no está diseñado para fuerzas que dependen explícitamente de la velocidad: en realidad no hay manera de evaluar de manera exacta la fuerza $\vec{F}(t+dt) = \vec{F}(\vec{v}(t+dt))$, pues depende de la velocidad en $t + dt$ que aún no se conoce. La aproximación que se propone aquí tiene fines didácticos y no corresponde con algoritmos probados que ofrezcan una mejor precisión (como referencia rápida se puede consultar la página http://es.scribd.com/doc/96319261/Algoritmo-de-Verlet).

viii. Esto es independiente del cambio de signo en cualquier componente de $\vec{v}$.

ix. La función `reboteP()` modifica posición y velocidad *solamente si* hay traslape de la esfera (partícula) con el plano.

x. Más adelante abordaremos el caso de muchas partículas.

xi. Las unidades las obviamos en este trabajo: son arbitrarias. Sin embargo sería bueno para los estudiantes establecer unidades e incluso hacer conversiones entre ellas.

xii. Hay que notar que la posición también requiere una corrección: cuando se pregunta si el centro de la partícula ya «cruzó» una pared, además de cambiar el signo de la velocidad hay que «regresar» a la partícula hacia dentro de la caja.

xiii. Dado que, para efectos gráficos, definimos el espacio donde las partículas pueden moverse a $2L - 2r$, donde r es el radio de las «partículas».

xiv. Si a este problema se le agregara un campo gravitacional uniforme, observaríamos que el centro de masa describe la trayectoria de un tiro parabólico, aun cuando las trayectorias de las dos partículas sean, en principio, más complejas.

xv. Es decir, una colisión en la cual los centros de masa se aproximan sobre la línea que los une. Una discusión amplia sobre colisiones elásticas se puede ver en la Ref. 9.

xvi. La expresión utilizada aquí puede encontrarse en cualquier libro de Mecánica elemental pero, es necesario aclarar, esta fórmula se obtiene a partir de una colisión frontal. En general las direcciones de las velocidades después de la colisión dependen de la dirección relativa en que se mueve cada una de las

partículas, sin embargo, creemos que una primera aproximación a estos tópicos puede ser de utilidad.

$xvii$. Discusiones sobre el tema deben abrir los horizontes de visión de los estudiantes hacia tópicos de actualidad como la Teoría del Caos o el estudio de estabilidad en Sistemas Dinámicos.

$xviii$. Donde se asume que las partículas son absolutamente indeformables y las interacciones entre ellas completamente elásticas.

xix. Como las fuerzas gravitacionales solamente dependen de la distancia se dice que poseen simetría radial o esférica.

xx. Trabajos como «Apliación del Modelo de Hardenberg para el Estudio Dinámico de Formación de Patrones de Vegetación», «Aplicación de los Sistemas Dinámicos en la solución de problemas de polución atmosférica», «Modelo de Crecimiento de Tumores Cancerígenos con Autómatas Celulares», «Investigación del efecto de inversión de carga en el problema de la doble capa eléctrica», «Formación de moléculas a partir de potenciales atractivos» y otros más han sido presentados en el congreso nacional de la SMF.

xxi. En estos eventos periódicos han participado algunos estudiantes de esta facultad trabajando diferentes tópicos de su interés principalmente.

1. C. Lanczos, *The variational principles of Mechanics* (University of Toronto Press, 1949).
2. S. T. Thornton and J. B. Marion, *Classical dynamics of particles and systems*. (Brooks/Cole, 5 edition, 2004).
3. M. P. Allen and D. J. Tildesley, *Computer Simulations of Liquids* (Clarendom Press, Oxford, 1987).
4. D. Frenkel and B. Smit, *Understanding Molecular Simulation. From Algorithms to Applications* (Academic Press, 1996).
5. J. F. Rojas, M. A. Morales, A. Rangel, and I. Torres, *Rev. Mex. Fis.*, **55** (2009) 97–111.
6. R. L. Burden and J. Douglas Faires, *Análisis Numérico* (International Thomson Editores, 6th edition, 1998).
7. M. F. Ferrerira da Silva, *Rev. Mex. Fis.*, **54** (2008) 65–74.
8. A.Ñ. Matvéev, *Física Molecular* (Ed. MIR, 1987).
9. S. Díaz-Solórzano and L. A. González-Díaz, *Rev. Mex. Fis.* **55** (2009) 57–60.
10. E. Lorenz, *The Essence of Chaos*. (University of Washington Press, 1993).
11. E.Ñ. Lorenz, *Journal of the Atmospheric Sciences*, **20** (1963) 130–141.