Cinemática del Punto Material

Ángel Piñeiro. Física I

September 25, 2022

Cinemática Caída Libre con Velocidad inicial $\vec{v}_0$

Objetivo: Describir el movimiento de una partícula en caída libre, es decir sometida a la aceleración gravitatoria, partiendo desde una plataforma con velocidad inicial $\vec{v}_0$.

Paso 1: Importamos las librerías que necesitaremos para realizar los cálculos, animaciones y representaciones gráficas. Definimos también una función para controlar un botón que controle la ejecución/pausa de los cálculos y la simulación:

[ ]:
```
from vpython import *
import numpy as np
import sympy as sp
sp.init_printing() # output formateado en latex
import sympy.physics.vector as spv
import IPython.display as disp

run = False
def runbutton(b): # Se llama a esta función cuando se hace click en el botón de␣
 ↪ejecutar
    global run
    if run: b.text = 'Ejecutar' # b es el botón
    else: b.text = 'Pausa'
    run = not run
```

Paso 2: Simularemos una trayectoria de caída libre:

$$\vec{a} = -9.8\,\hat{j}\,m/s^2$$

asumiendo que el eje y es el eje vertical) para una partícula que parte de una velocidad inicial predefinida $\vec{v}_0$. En este caso no necesitamos cálculo simbólico ya que la ecuación de la trayectoria es bien conocida:

$$\vec{r}(t) = \vec{r}_0 + \vec{v}_0\,t + 1/2\,\vec{a}\,t^2$$

Necesitamos entonces definir la posición y velocidad inicial de la partícula, así como el vector aceleración:

[ ]:
```
r0=vector(-14,-5,0)
v0=vector(2,20,0)
a=vector(0,-9.806,0)
```

Paso 3: Para hacer la simulación vamos a definir una esfera que representará la partícula y una plataforma desde la cuál ésta iniciará el movimiento. Asociamos a la esfera un vector velocidad y un vector aceleración para representarlos en la simulación de manera dinámica. Incluimos también el botón de ejecución llamando a la función que definimos al principio del programa.

Definimos también gráficas en las que representaremos las componentes x e y de las posiciones y velocidades de la partícula, su trayectoria y su hodógrafa (la gráfica de las componentes de la velocidad).

Se inicializa el tiempo a 0 y se ejecuta la simulación calculando la posición y la velocidad cada 0.005 segundos. La simulación termina cuando la esfera choca contra la plataforma o cuando la componente x de la partícula es mayor que la longitud de la plataforma. Las gráficas se dibujan a tiempo real.

OJO!! aquí es donde introducimos la ecuación de la trayectoria!:

[ ]:
```
escena1 = canvas(background=color.white, autoscale=False, width=1000,height=300) #␣
 ↪Escenario donde se realiza la simulación
bola = sphere(canvas=escena1, pos=r0, radius=0.4, color=color.red, make_trail=True,␣
 ↪emissive=True, align='left')
bola.v=v0 # asociamos un vector "v" a la esfera
bola.a=a # asociamos un vector "a" a la esfera

attach_arrow(bola, 'v', scale=0.5, color=color.blue) # vector velocidad en azul para␣
 ↪representación
attach_arrow(bola, 'a', scale=0.5, color=color.yellow) # vector aceleración en␣
 ↪amarillo para representación

posicionbase=vector(0,-5,0) # localización de la plataforma (coordenadas de su centro␣
 ↪geométrico)
tamanhobase=vector(30,0.2,10) # Tamaño de la plataforma en las 3 dimensiones
base = box(canvas=escena1, pos=posicionbase, size=tamanhobase, texture=textures.wood,␣
 ↪align='left') # Asociamos la plataforma a la variable "base"

button(text='Ejecutar', bind=runbutton) # Dibujamos el botón de ejecución

# Definimos los gráficos que representaremos
posiciones=graph(title='Posición de la partícula', width=600, height=200,
                 xtitle='<i>t</i>', ytitle='<i>Posición</i>', align='left')
trayectoria=graph(title='Trayectoria de la partícula', width=300, height=200,
                  xtitle='<i>x</i>', ytitle='<i>y</i>', align='right')
velocidades=graph(title='Velocidad de la partícula', width=600, height=200,
                  xtitle='<i>t</i>', ytitle='<i>Velocidad</i>', align='left')
hodografa=graph(title='Hodógrafa de la partícula', width=300, height=200,
                xtitle='<i>v<sub>x</sub></i>', ytitle='<i>v<sub>y</sub></i>',␣
 ↪align='right')

x = gcurve(color=color.black, legend=True, label="x", graph=posiciones)
y = gcurve(color=color.red, legend=True, label="y", graph=posiciones)
trj=gcurve(color=color.black, graph=trayectoria)
vx = gcurve(color=color.black, legend=True, label="<i>v<sub>x</sub></i>",␣
 ↪graph=velocidades)
vy = gcurve(color=color.red, legend=True, label="<i>v<sub>y</sub></i>",␣
 ↪graph=velocidades)
hod=gcurve(color=color.black, graph=hodografa)

t=0; dt=0.005 # inicializamos el tiempo y el paso de tiempo

# calculamos la posición y velocidad a intervalos de tiempo definidos por la variable␣
 ↪dt
while bola.pos.x < tamanhobase.x*0.5 and bola.pos.y > posicionbase.y-0.1: #␣
 ↪condiciones para parar la simulación
    rate(50) # iteraciones por segundo
```

```
    if run:
        bola.v=bola.v+a*dt # Actualizamos la velocidad
        bola.pos=bola.pos+bola.v*dt+0.5*a*dt**2 # Ecuación de la trayectoria para un␣
↪movimiento de caída libre
        y.plot(t, bola.pos.y-r0.y) # Graficamos la componente y de la posición
        x.plot(t, bola.pos.x-r0.x) # Graficamos la componente x de la posición
        trj.plot(bola.pos.x,bola.pos.y) # Gráfica de la trayectoria:x vs y
        vx.plot(t, bola.v.x) # Graficamos la componente x de la velocidad
        vy.plot(t, bola.v.y) # Graficamos la componente y de la velocidad
        hod.plot(bola.v.x, bola.v.y) # Hodógrafa: vx vs vy
        t=t+dt
        if bola.pos.y < posicionbase.y-0.1: # si choca con la plataforma rebota
            bola.v.y=np.abs(bola.v.y) # rebote elástico
            bola.pos.y=posicionbase.y # para que no se quede atrapado en el condicional
```