Cinemática del Punto Material

Ángel Piñeiro. Física I

September 24, 2022

Cinemática Movimiento Circular Plano

Objetivo: obtener la velocidad y la aceleración de una partícula que describe una trayectoria circular, expresada en forma paramétrica en función del tiempo.

Input: Plantearemos el problema de manera general a través de una función vectorial, con sus componentes x, y, z, parametrizada en función del tiempo:

$$\vec{r}(t) = x(t)\,\hat{i} + y(t)\,\hat{j} + z(t)\,\hat{k}$$

Procedimiento: Calcularemos la velocidad y la aceleración de la partícula, así como sus componentes intrínsecas, a través de las derivadas de la función $\vec{r}(t)$:

$$\text{Velocidad:} \quad \vec{v}(t) = \frac{d\vec{r}(t)}{dt} = \dot{\vec{r}}(t)$$

$$\text{Aceleración:} \quad \vec{a}(t) = \frac{d\vec{v}(t)}{dt} = \dot{\vec{v}}(t) = \ddot{\vec{r}}(t)$$

$$\text{Vector unitario tangencial:} \quad \hat{\tau}(t) = \frac{\vec{v}(t)}{|\vec{v}(t)|}$$

$$\text{Aceleración tangencial:} \quad \vec{a}_\tau(t) = (\vec{a} \cdot \hat{\tau}) \cdot \hat{\tau}$$

$$\text{Aceleración normal:} \quad \vec{a}_n(t) = \vec{a}(t) - \vec{a}_\tau(t)$$

$$\text{Vector unitario normal:} \quad \hat{n}(t) = \frac{\vec{a}_n(t)}{|\vec{a}_n(t)|}$$

Resolveremos el problema de manera general utilizando cálculo simbólico en Python. De esta manera el código será válido para cualquier tipo de trayectoria con solo cambiar la función $\vec{r}(t)$. Incluiremos también diferentes representaciones gráficas y simulaciones animadas de los movimientos para ayudar a entender el problema.

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 = 'Iniciar' # b es el botón
    else: b.text = 'Pausa'
    run = not run
```

Paso 2: Definimos la amplitud y frecuencia del movimiento circular, así como tiempo máximo de nuestra simulación.

Definimos la ecuación de la trayectoria parametrizada en función del tiempo, como un movimiento circular en el plano XY. Para ello necesitamos definir primero el símbolo t (tiempo) y un sistema de referencia (R) que nos permitirá utilizar una base de vectores unitarios en la dirección de los ejes de coordenadas

[ ]:
```
A=5; w=2  # parámetros del movimiento circular: amplitud y frecuencia
tmax=8; # tiempo máximo de nuestra simulación

t = sp.symbols('t') # Definimos el símbolo t para representar el tiempo
R= spv.ReferenceFrame('R') # Definimos un sistema de coordenadas para representar los␣
↪vectores

r_t=A*(-sp.cos(w*t)*R.x+sp.sin(w*t)*R.y) # Aquí definimos nuestra trayectoria

disp.display(disp.Math(r'\vec{r}(t)= '), r_t) # Visualizamos la ecuación de la␣
↪trayectoria
```

[ ]:
```
v_t=r_t.diff(t,R) # derivada en cálculo simbólico para obtener la velocidad a partir␣
↪de la posición
a_t=v_t.diff(t,R) # derivada en cálculo simbólico para obtener la aceleración a partir␣
↪de la velocidad

disp.display(disp.Math(r'\vec{v}(t)= '), v_t) # Visualizamos la velocidad
disp.display(disp.Math(r'\vec{a}(t)= '), a_t) # Visualizamos la aceleración
```

La velocidad siempre debe ser tangente a la posición y por tanto ambos vectores deben ser perpendiculares, por lo que su producto escalar es nulo:

[ ]:
```
spv.dot(r_t, v_t)
```

Si queremos un vector unitario tangente a la trayectoria ($\hat{\tau}$), sólo tenemos que dividir la velocidad entre su módulo:

[ ]:
```
tau_t=v_t/v_t.magnitude()
disp.display(disp.Math(r'\hat{\tau}(t)= '),tau_t)
```

La componente tangencial de la aceleración se obtiene proyectando la aceleración sobre el vector $\hat{\tau}$. En un movimiento circular uniforme la componente tangencial de la aceleración debe ser nula:

[ ]:
```
atau_t=spv.dot(a_t,tau_t)*tau_t
disp.display(disp.Math(r'\vec{a}_{\tau}(t)= '),atau_t)
```

La componente normal de la aceleración se obtiene como la diferencia entre la aceleración y su componente tangencial:

[ ]:
```
atau_n=a_t-atau_t
disp.display(disp.Math(r'\vec{a}_{n}(t)= '),atau_n)
```

Paso 3: Para hacer la simulación vamos a definir una esfera que representará la partícula, a al cual asociaremos 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).

[ ]:
```
escena1 = canvas(background=color.white, autoscale=False, width=1000,height=300) #␣
 ↪Escenario donde se realiza la simulación

tiempo=0 # inicializamos el tiempo

# a continuación definimos las componentes x e y de cada magnitud en el instante␣
 ↪inicial
posx=spv.dot(r_t, R.x).evalf(subs={t: tiempo})
posy=spv.dot(r_t, R.y).evalf(subs={t: tiempo})
velx=spv.dot(v_t, R.x).evalf(subs={t: tiempo})
vely=spv.dot(v_t, R.y).evalf(subs={t: tiempo})
acx=spv.dot(a_t, R.x).evalf(subs={t: tiempo})
acy=spv.dot(a_t, R.y).evalf(subs={t: tiempo})

# vectores iniciales
r0=vector(posx, posy, 0)
v0=vector(velx,vely,0)
a0=vector(acx,acy,0)

# definimos la esfera y los vectores v y a
bola = sphere(pos=r0, radius=0.4, color=color.red, make_trail=True, emissive=True)
bola.v=v0 # asociamos un vector "v" a la esfera
bola.a=a0 # asociamos un vector "a" a la esfera
attach_arrow(bola, 'v', scale=0.25, color=color.blue) # vector velocidad en azul para␣
 ↪representación
attach_arrow(bola, 'a', scale=0.25, color=color.yellow) # vector aceleración en␣
 ↪amarillo para representación

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

# definimos las gráficas que representaremos
posiciones=graph(title='Posición de la partícula', width=600, height=200,
                 xmin=0, xmax=tmax, ymin=-A, ymax=A, align='left',
                 xtitle='<i>t</i>', ytitle='<i>Posición</i>')
trayectoria=graph(title='Trayectoria de la partícula', width=300, height=200,
                  xmin=-A, xmax=A, ymin=-A, ymax=A, align='right',
                  xtitle='<i>x</i>', ytitle='<i>y</i>')
velocidades=graph(title='Velocidad de la partícula', width=600, height=200,
                  xmin=0, xmax=tmax, ymin=-A*w, ymax=A*w, align='left',
                  xtitle='<i>t</i>', ytitle='<i>Velocidad</i>')
hodografa=graph(title='Hodógrafa de la partícula', width=300, height=200,
                xmin=-A*w, xmax=A*w, ymin=-A*w, ymax=A*w, align='right',
                xtitle='<i>v<sub>x</sub></i>', ytitle='<i>v<sub>y</sub></i>')
```

```
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)

# calculamos la posición y velocidad a intervalos de tiempo definidos por la variable␣
 ↪dt
while (tiempo <tmax): # condiciones para parar la simulación
    rate(10) # iteraciones por segundo
    if run:
        tiempo=tiempo+0.02 # Actualizamos el tiempo
        posx=spv.dot(r_t, R.x).evalf(subs={t: tiempo}) # componente x de la posición
        posy=spv.dot(r_t, R.y).evalf(subs={t: tiempo}) # componente y de la posición
        velx=spv.dot(v_t, R.x).evalf(subs={t: tiempo}) # componente x de la velocidad
        vely=spv.dot(v_t, R.y).evalf(subs={t: tiempo}) # componente y de la velocidad
        acx=spv.dot(a_t, R.x).evalf(subs={t: tiempo}) # componente x de la aceleración
        acy=spv.dot(a_t, R.y).evalf(subs={t: tiempo}) # componente y de la aceleración
        bola.a=vector(acx,acy,0) # Actualizamos la aceleración
        bola.v=vector(velx,vely,0) # Actualizamos la velocidad
        bola.pos=vector(posx,posy,0) # Actualizamos la posición
        x.plot(tiempo, posx) # Graficamos la componente x de la posición
        y.plot(tiempo, posy) # Graficamos la componente y de la posición
        trj.plot(posx,posy) # Gráfica de la trayectoria: x vs y
        vx.plot(tiempo, velx) # Graficamos la componente x de la velocidad
        vy.plot(tiempo, vely) # Graficamos la componente y de la velocidad
        hod.plot(velx,vely) # Hodógrafa: vx vs vy
```

Paso 4: Prueba a cambiar los parámetros del movimiento en el Paso 2 (amplitud, frecuencia o tiempo total de la simulación) para ver cómo afectan a la trayectoria.

Puedes cambiar también la ecuación de la trayectoria en el Paso 2 haciendo que no sea circular.