Parcial 2 Simulación

Punto 1.

Modificaciones hechas en la simulación 1 de caída libre:
Original:

In [1]:
```
#################################################################
# Ejemplo de creación de un Sistema generador de partículas aleatorias
# que se Lanzan en tiro parabólico con rebote
# Universidad Militar Nueva Granada
# Ingeniería en Multimedia Campus
# Simulación 2021-02
# Ing. Eduard Sierra
#################################################################
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from numpy import sin, cos, pi, outer, ones, size, linspace
import numpy as np
from PIL import Image
import random
```

Modificación:

In [508]:
```
#################################################################
# Parcial 2do Corte Simulación
# Universidad Militar Nueva Granada
# Ingeniería en Multimedia Campus
# Simulación 2022-02
# Gustavo Leon
#################################################################
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from numpy import sin, cos, pi, outer, ones, size, linspace
import numpy as np
from PIL import Image
import random
```

Original:

```
a=Vector3D(3,2,1)
b=Vector3D(1,2,3)
c=Vector3D(0,0,0)
print(c)
c=a+b
b=a-a
a=a*3
#c.x=a.x+b.x
#c.y=a.y+b.y
#c.z=a.z+b.z

print(a.x,a.y,a.z)
print(b.x,b.y,b.z)
print(c.x,c.y,c.z)

```

Modificación: Se eliminó esta sección

Original:

```
class Particula:
    def __init__(self):
        self.p=Vector3D(0,0,0)
        self.v=Vector3D(0,0,0)
        self.f=Vector3D(0,0,0)
        self.m=0
        self.color=(0,0,0)
        self.tam=1

    def cambiarPosicion(self,pos):
        self.p=pos

    def cambiarVelocidad(self,vel):
        self.v=vel

    def cambiarFuerza(self,fuerza):
        self.f=fuerza

    def cambiarMasa(self,masa):
        self.f=m
    def cambiarTamanio(self,tam):
        self.tam=tam
    def cambiarColor(self,color):
        self.color=color

    def __str__(self):
        return 'Posicion \t%s \nVelocidad \t%s \n' % (self.p,self.v)

    def __repr__(self):
        return 'Posicion \t%s \nVelocidad \t%s \n' % (self.p,self.v)

    def gParticula(self):
        #Generación de puntos de geometria paramétrica de una esfera
        #el centro está dado por
        #el tamaño del radio se tomará r=self.tam
        r=self.tam
        a = linspace(0, 2*pi,int(r*5))
        b = linspace(0, pi,int(r*5))
        xc = self.p.x+r * outer(cos(a), sin(b))
        #por razones de visualización para evitar deformación
        #con esta perspectiva se ve mas circular
        #en tmaños de despliegue mayores a 100
        #se cambio la ecuación parametrica
        # debia ser yc = self.p.y+r * outer(sin(a), sin(b))
        yc = self.p.y+0.3*r * outer(sin(a), sin(b))
        zc = self.p.z+r * outer(ones(size(a)), cos(b))
        return xc,yc,zc

```

Modificación: Se agregó cambiarRestitucion para la posterior modificación del coeficiente de restitución.

```
class Particula:
    def __init__(self):
        self.p=Vector3D(0,0,0)
        self.v=Vector3D(0,0,0)
        self.f=Vector3D(0,0,0)
        self.m=1
        self.color=(0,0,0)
        self.tam=1
        self.kRestitucion=0

    def cambiarPosicion(self,pos):
        self.p=pos

    def cambiarVelocidad(self,vel):
        self.v=vel

    def cambiarFuerza(self,fuerza):
        self.f=fuerza

    def cambiarMasa(self,masa):
        self.f=m

    def cambiarTamanio(self,tam):
        self.tam=tam

    def cambiarColor(self,color):
        self.color=color

    def cambiarRestitucion(self,kRestitucion):
        self.kRestitucion=kRestitucion

    def __str__(self):
        return 'Posicion \t%s \nVelocidad \t%s \n' % (self.p,self.v)

    def __repr__(self):
        return 'Posicion \t%s \nVelocidad \t%s \n' % (self.p,self.v)

    def gParticula(self):
        #Generación de puntos de geometria paramétrica de una esfera
        #el centro está dado por
        #el tamaño del radio se tomará r=self.tam
        #por razones de visualización para evitar deformación
        #con esta perspectiva se ve mas circular
        #en tmaños de despliegue mayores a 100
        #se cambio la ecuación parametrica
        # debia ser yc = self.p.y+r * outer(sin(a), sin(b))
        r=self.tam
        a = linspace(0, 2*pi,int(r*5))
        b = linspace(0, pi,int(r*5))
```

Original:

```
from matplotlib.colors import LightSource, Normalize
from matplotlib.colors import LinearSegmentedColormap
part=Particula()
p=Vector3D(1,2,3)
part.cambiarPosicion(p)
part.cambiarTamanio(20)

rgb=(1,0.5,1)
part.cambiarColor(rgb)
print(part)
part.p=Vector3D(1,2,30)
print(part)
x,y,z = part.gParticula()
fig = plt.figure(figsize=(10, 10))

ax = fig.add_subplot(1,1,1, projection='3d')
ax.plot_surface(x, z, y, color=part.color, shade=False)
#ax.scatter(part.p.x, part.p.z, part.p.y, color=('b'))

#ax.scatter(part.p.x, part.p.z, part.p.y, color=rgb)
ax.set_xlim(-100,100)
ax.set_ylim(-100,100)
ax.set_zlim(-100,100)

```

Modificación:

```
from matplotlib.colors import LightSource, Normalize
from matplotlib.colors import LinearSegmentedColormap
part=Particula()

```

Original:

```
class SistemaParticulasAleatorias:
    def __init__(self,n):
        self.particulas=[]
        self.n=n
        self.t=0
        self.gravedad=-9.8
        self.kRestitucion=0.8
        self.paso=0.1
        for i in range(0,int(n)): #en C o java for(i=0;i<n;i++)
            paux=Particula()
            self.particulas.append(paux)
        #print(self.particulas)

    def CreacionAleatoria(self):
        for part in self.particulas:
            #Posicion inicial aleatroai saleido de y=0, en el espacio de x en (0,50) y z en (0,50)
            part.cambiarPosicion(Vector3D(random.randint(-10,10),0,random.randint(-10,10)))

            #Cinco tamaños diferentes
            t=random.randint(1,10)
            part.cambiarTamanio(t)

            #colores
            color=(random.random(),random.random(),random.random())#Usa random apra genrar valores entre 0 y 1 para los valo
            part.cambiarColor(color)

            #Velocidades iniciales hacia 4 direcciones diferentes
            r=random.randint(0,3)
            if(r==0):
                part.cambiarVelocidad(Vector3D(-random.randint(0,20),random.randint(0,50),random.randint(0,20)))
            if(r==1):
                part.cambiarVelocidad(Vector3D(random.randint(0,20),random.randint(0,50),-random.randint(0,20)))
            if(r==2):
                part.cambiarVelocidad(Vector3D(-random.randint(0,20),random.randint(0,50),-random.randint(0,20)))
            if(r==3):
                part.cambiarVelocidad(Vector3D(random.randint(0,20),random.randint(0,50),random.randint(0,20)))
            part.m=1
        #print(self.particulas)

```

Modificación: Se agrega el modificador de kRestitucion en SistemasParticulasAleatorias.
Se agrega un contador en CreacionAleatoria, para usar dentro de un condicional en el cual se modifican las físicas de las partículas creadas.

```
class SistemaParticulasAleatorias:
    def __init__(self,n):
        self.particulas=[]
        self.n=n
        self.t=0
        self.gravedad=-9.8
        self.paso=0.1

        for i in range(0,int(n)): #en C o java for(i=0;i<n;i++)
            paux=Particula()
            self.particulas.append(paux)
        print(self.particulas)

    #Modificar condiciones iniciales aqui.
    def CreacionAleatoria(self):
        cont=0
        for part in self.particulas:

            if(cont==0):
                part.cambiarPosicion(Vector3D(130,80, 200))
                t=random.randint(18,25)
                part.cambiarTamanio(t)
                color=(random.random(),random.random(),random.random())
                part.cambiarColor(color)
                part.cambiarVelocidad(Vector3D(15,20,0))
                part.cambiarRestitucion(1)
            if(cont==1):
                part.cambiarPosicion(Vector3D(-100,40,150))
                t=random.randint(24,35)
                part.cambiarTamanio(t)
                color=(random.random(),random.random(),random.random())
                part.cambiarColor(color)
                part.cambiarVelocidad(Vector3D(15,20,2))
                part.cambiarRestitucion(0.5)
            if(cont==2):
                part.cambiarPosicion(Vector3D(160,60,180))
                t=random.randint(15,18)
                part.cambiarTamanio(t)
                color=(random.random(),random.random(),random.random())
                part.cambiarColor(color)
                part.cambiarVelocidad(Vector3D(20,15,5))
                part.cambiarRestitucion(0)
            part.m=1
            cont=cont+1

        print(self.particulas)
```

Original:

```
def PasoSimulacion(self,n):

    ##################################################
    #Aplicar fuerzas y actualizar estado de fase usando método de Euler
    ##################################################

    for part in self.particulas:
        #Actualizar posición
        part.p=part.p+self.dp(self.t,part.p,part.v)*self.paso
        #Actualizar velocidad
        part.v=part.v+self.dv(self.t,part.p,part.v,part.m)*self.paso

        #Verifica colisión con el piso
        #Ecuación del plano Ax+By+Cz+D=0 para el suelo 0x+1y+0z+0=0 => y=0
        #Vector normal (A,B,C)==>(0,1,0)
        #Evalúa ecuación del plano con los valores de las coordenadas del punto
        #El punto esté en el plano si al evaluar la ecuación esta da
        y=part.p.y#evalua la ecuación
        #verifica colision si el signo es menor entre el producto punto (X-P)N
        #Hay colision exacta cuando este prducto es igual a cero
        #(X posición de la particula)
        #(P posición de un punto en el plano, N, normal del plano)
        if(y<=0):#Con la desigualdad verifica el signo, con la igualdad el contacto exacto
            part.v.y=-self.kRestitucion*part.v.y #Cambia la dirección de la velocidad, para cambiar la dirección de movi




    #Actualizar tiempo
    self.t=self.t+paso



    self.Graficar(n)
```

Modificación:Se modifico PasoSimulacion agregando la función del plano, junto a los valores de la posición de la partícula, agregando un condicional para la colisión con el nuevo plano creado, se agrega la función de RuggenKutta orden 4 para hacer pruebas con ambos métodos.

```
def PasoSimulacion(self,n):

    ################################################
    #Aplicar fuerzas y actualizar estado de fase usando método de Euler
    ################################################

    for part in self.particulas:
        #Actualizar posición
        part.p=part.p+self.dp(self.t,part.p,part.v)*self.paso
        #Actualizar velocidad
        part.v=part.v+self.dv(self.t,part.p,part.v,part.m)*self.paso

        #A cada variable se le asigna el valor de la posicion de la particula
        y=part.p.y
        x=part.p.x
        z=part.p.z
        p=-0.5*(x)+y+0.25*(z)-1

        if(y<=-1): #Evaula colisión con el suelo
            part.v.y=-part.kRestitucion*part.v.y

        if (p<=1): #Evalua colisión con el plano
            part.v.y=-part.kRestitucion*part.v.y


    #Actualizar tiempo
    self.t=self.t+paso
    self.Graficar(n)

def Rungekutta4(x0,y0,xf,h):
    tam=((xf-xi)/h)+1
    x=np.zeros(int(tam))
    y=np.zeros(int(tam))
    x[0]=x0
    y[0]=y0
    for i in range(int(tam)-1):

        k1=diferencial(x[i],y[i])
        k2=diferencial(x[i]+h/2, y[i]+(h/2)*k1)
        k3=diferencial(x[i]+h/2, y[i]+(h/2)*k2)
        k4=diferencial(x[i]+h/2,y[i]+h*k3)
        y[i+1]=y[i]+(h/6)*(k1+2*k2+2*k3+k4)
        x[i+1]=x[i]+h
        print(x)
        print(y)
    return x,y
```

Original:

```
def Simular(self, duracion,paso):
    self.paso=paso
    frames = int(((duracion)/paso)+1)
    self.t=0;
    self.Graficar(0)#Grafica t=0
    print('Cuadro '+str(0)+' Finalizado')
    for n in range(1,frames):
        self.PasoSimulacion(n)
        print('Cuadro '+str(n)+' Finalizado')



    images = [Image.open(f"Secuencia/SPA{n}.png") for n in range(frames)]
    images[0].save('Secuencia/SPAFinal.gif', save_all=True, append_images=images[1:], duration=100, loop=0)
    print('Terminé')


def Graficar(self,n):
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(1,1,1, projection='3d')

    #Ciclo para graficar todas las particulas
    for part in self.particulas:
    #for part in range(2):
        #1. Grafica con esferas de diferentes tamañosprint(part.tam)
        #genera la geometría de la particula como una esfera
        x,y,z = part.gParticula()
        #Grafica la esfera de la particula como una superficie
        #con shade=False, queda de color plano sin sombra
        ax.plot_surface(x, z, y, color=part.color, shade=True)

        #2. Activar para graficar con puntos
        #ax.scatter(part.p.x, part.p.z, part.p.y, color=part.color)#grafica como puntos

    ax.set_xlim(-500,500)
    ax.set_ylim(-500,500)
    ax.set_zlim(0,100)
    plt.title("t = "+str(round(self.t,2)))
    plt.savefig(f"Secuencia/SPA{n}.png")
    #plt.show()
    plt.close()
```

Modificaciones: Se agrega la función def Plano en el cual se hace la creación del plano adicional que tocaba realizar. Y en la función graficar, se agrega los valores iniciales del plano a realizar, junto con la gratificación de este.

```
def Simular(self, duracion,paso):
    self.paso=paso
    frames = int(((duracion)/paso)+1)
    self.t=0;
    self.Graficar(0)#Grafica t=0
    print('Cuadro '+str(0)+' Finalizado')
    for n in range(1,frames):
        self.PasoSimulacion(n)
        print('Cuadro '+str(n)+' Finalizado')

    images = [Image.open(f"Secuencia/SPA{n}.png") for n in range(frames)]
    images[0].save('Secuencia/SPAFinal.gif', save_all=True, append_images=images[1:], duration=100, loop=0)
    print('Terminé')

def Plano (self,a,b,c,d,x1,y2):
    A = np.array([[a, b, c]])
    z = lambda x,y: (A[0,0]*x + A[0,1]*y -d)
    x = x1
    y = y2
    X,Y=np.meshgrid(x, y)
    Z = z(X,Y)

    return X, Y,Z

def Graficar(self,n):
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(1,1,1, projection='3d')
    a1=np.linspace(0,600,n)
    c1=np.linspace(0,800,n)
    a,b,c=self.Plano(-0.5,0.25,-1,1,a1,c1)

    #Ciclo para graficar todas las particulas
    for part in self.particulas:
    #for part in range(2):
        #1. Grafica con esferas de diferentes tamañosprint(part.tam)
        #genera la geometría de la particula como una esfera
        x,y,z = part.gParticula()
        #Grafica la esfera de la particula como una superficie
        #con shade=False, queda de color plano sin sombra
        ax.plot_surface(x, z, y, color=part.color, shade=True)
        #2. Activar para graficar con puntos
        #ax.scatter(part.p.x, part.p.z, part.p.y, color=part.color)#grafica como puntos

    ax.view_init(30,10)
    ax.set_xlim(-50,500)
    ax.set_ylim(-300,500)
    ax.set_zlim(-150,150)
    ax.plot_surface(a,b,c, color=('b'),alpha=0.1)
    plt.title("t = "+str(round(self.t,2)))
    plt.savefig(f"Secuencia/SPA{n}.png")
```

Punto 3.

Para este punto se hicieron exactamente las mismas modificaciones que en el primer punto, pero en dentro de creación partícula en los condicionales, se modifica el vector velocidad para hacer el efecto de tiro parabólico.