PROYECTO 1 - Laboratorio de Simulación

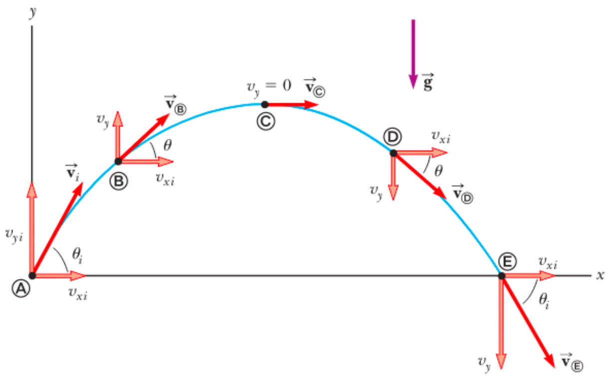
Ricardo Escobar Matzir 202002342, Marcelo Lam 201903603

27 de abril de 2024

1. Problema 1

1.1. Tiro parabólico

La implementanción tendrá como objetivo obtener el alcance horizontal, altura máxima y tiempo de vuelo de un atleta que participa en salto de longitud y deja el suelo a un ángulo de 20° sobre la horizontal y con una rapidez de 11 m/s.



Recuperado de: Física para ciencias e ingeniería con Física Moderna
Volumen 1. 7th. Edición. Raymond A. Serway, John W. Jewett, Jr.

Las expresiones son las siguientes:

- Tiempo en llegar a altura máxima:
$$t = \frac{v_i \sin\theta_i}{g}$$
- Tiempo de vuelo:
$$t_{total} = \frac{2v_i \sin\theta_i}{g} = 2t$$
- Alcance horizontal:
$$\frac{v_i^2 \sin(2\theta_i)}{g} = 2v_i \cos\theta_i \cdot t$$
- Altura máxima:
$$h = \frac{v_i^2 \sin^2\theta_i}{2g} = \frac{v_i \sin\theta_i}{2} \cdot t$$

Notar que el alcance horizontal, la altura máxima y el tiempo de vuelo pueden dejarse en función del tiempo t, esto nos ayudará a crear una **clase padre** "Tiempo"que heredará el método “calculotiempo"para que las clases hijas puedan tomar el valor de tiempo t sin tener que calcularlo otra vez.

1.2. Implementación en Python 3

A continuación, se comparte el enlace directo al ejercicio en Google Colab:

https://colab.research.google.com/drive/1rdow-iC4HnwWdqY7T8oo08vkNiZe9oJt?usp=sharing

- Clase padre:

```python
import math
conv = math.pi/180

#Clase padre:
class Tiempo:
  def __init__(self, vi:float, ang:float, grav:float):
    self.velocidadinicial= vi #Velocidad inicial en m/s.
    self.anguloincial= ang #Angulo en grados sexagesimalees.
    self.gravedad= grav #Gravedad en m/s^2.
  def calculotiempo(self):
    #Aqui se calcula el tiempo en alcanzar la altura maxima.
    t_medios = (self.velocidadinicial*math.sin(self.anguloincial*conv))/self.gravedad
    return t_medios

t = Tiempo(11,20,9.8)
print("Tiempo en alcanzar altura maxima:",round(t.calculotiempo(),2),"segundos")
```

```
Tiempo en alcanzar altura m xima: 0.38 segundos
```

- Primera clase hija para el **tiempo de vuelo**:

```python
class TiempoTotal(Tiempo):
    pass
    def tiempototal(self):
    t_total = 2*self.calculotiempo()
    return t_total
    def respuesta(self):
    print("El tiempo total de vuelo:",round(tt.tiempototal(),2),"segundos")

tt = TiempoTotal(11,20,9.8)
tt.respuesta()

```

```
Tiempo total de vuelo: 0.77 segundos
```

- Segunda clase hija para la **altura máxima**:

```python
class Altura(Tiempo):
    pass
    def alturamaxima(self):
      ymax = (1/2)*self.velocidadinicial*(math.sin(self.anguloincial*conv))*self.
    calculotiempo()
      return ymax
    def respuesta(self):
      print("La altura m xima es",round(y.alturamaxima(),2),"metros")

y = Altura(11,20,9.8)
y.respuesta()
```

```
La altura m xima es 0.72 metros
```

- Tercera clase hija para el **alcance horizontal**:

```
class Alcance(Tiempo):
    pass
    def alcance(self):
    x = 2*self.velocidadinicial*math.cos(self.anguloincial*conv)*self.calculotiempo()
    return x
    def respuesta(self):
      print("El alcance horizontal es:",round(d.alcance(),2),"metros")

d = Alcance(11,20,9.8)
d.respuesta()
```

```
El alcance horizontal es: 7.94 metros
```

- Aplicación del concepto de **polimorfismo** para enunciar los resultados de las clases hijas:

```
y = Altura(11,20,9.8)
d = Alcance(11,20,9.8)
tt = TiempoTotal(11,20,9.8)
for z in (y,d,tt):
  z.respuesta()
```

```
La altura m xima es 0.72 metros
El alcance horizontal es: 7.94 metros
El tiempo de vuelo es: 0.77 segundos
```

- Aplicación del concepto de **composición** para calcular el tiempo de vuelo:

```
class TiempoTotal:
    def __init__(self, vi:float, ang:float, grav:float):
          self.velocidadinicial= vi
          self.anguloincial= ang
          self.gravedad= grav
          self.tiempot = Tiempo(vi,ang,grav)
    def tiempototal(self):
      t_total = 2*self.tiempot.calculotiempo()
      return t_total

tt = TiempoTotal(11,20,9.8)
print("Tiempo total de vuelo:",round(tt.tiempototal(),2), "segundos")
```

```
Tiempo total de vuelo: 0.77 segundos
```

2. Problema 2

En este problema se resuelve primero el sistema estático, mediante la segunda ley de Newton se hallan las tensiones de las cuerdas. Se debe cumplir que

$$\sum \vec{F} = 0$$

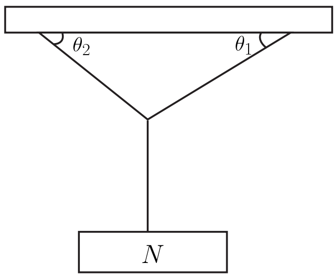


Figura 1: Sistema estático

y en componentes tenemos

$$\begin{cases} \sum F_x = T_1 \cos\theta_1 - T_2 \cos\theta_2 = 0 \\ \sum F_y = T_1 \sin\theta_1 + T_2 \sin\theta_2 - N = 0 \end{cases}$$

donde N es el peso de la caja en Newtons. Despejando para T_1 y T_2 obtenemos

$$T_1 = \frac{\kappa}{\cos\theta_1}, \qquad T_2 = \frac{\kappa}{\cos\theta_2} \tag{1}$$

donde

$$\kappa = \frac{N}{\tan\theta_1 + \tan\theta_2}$$

Luego implementamos esto a código para calcular estas tensiones a partir únicamente de los ángulos iniciales.

1. Primero creamos la clase y el constructor. En dicho constructor pedimos como parámetros los dos ángulos inciales que son `a1` y `a2`.

```
import math
class sis_estatico:
  def __init__(self, a1 = 0, a2 = 0, N = 0):
    self.a1 = a1
    self.a2 = a2
    self.N = N
    self.tensiones = dict()
    self.tensiones_calc()
```

2. Luego en la última linea llamamos al método `tensiones_calc()` para realizar el cálculo de las tensiones desde un inicio.

3. Seguido a esto creamos el método de `tensiones_calc()` que recién mencionamos donde aplicamos las ecuaciones (1) y lo guardamos en un diccionario.

```
def tensiones_calc(self):
    conv = math.pi/180
    k = (self.N)/(math.tan(self.a1*conv)+math.tan(self.a2*conv))
    T1 = k/math.cos(self.a1*conv)
    T2 = k/math.cos(self.a2*conv)
    self.tensiones["Tension 1"] = T1
    self.tensiones["Tension 2"] = T2
    return self.tensiones
```

4. Por último, creamos una instancia para probar el funcionamiento

```
sistema1= sis_estatico(a1 = 25,
                       a2 = 60,
                       N = 490)
```

lo cual nos da como resultado

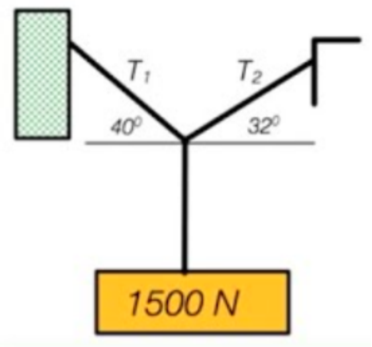
```
sistema1.tensiones
{'Tension 1': 245.93586019812017, 'Tension 2': 445.7871704182263}
```

resultado que podemos verificar manualmente.

3. Problema 3

3.1. Replanteamiento del problema

Para el problema anterior debíamos encontrar las tensiones T_1 y T_2 para la siguiente figura:



Para este problema debemos agregar más cuerdas al sistema con una tensión y ángulo conocidos, tomaremos unos detalles en cuenta:

- Llamemos a las tensiones de las cuerdas agregadas T_n, las cuerdas para un n impar serán agregadas al lado **izquierdo** del peso, para las n par serán agregadas al lado **derecho** del peso.
- Para todas las cuerdas agregadas, la suma de sus componentes horizontales será S_x y la suma de sus componentes verticales S_y. De la siguiente manera

$$S_x = T_3 \cos a_3 + T_4 \cos a_4 + ... + T_n \cos a_n$$

$$S_y = T_3 \sin a_3 + T_4 \sin a_4 + ... + T_n \sin a_n$$

- Para las cuerdas agregadas con n impar, el ángulo a_n debe de ser $180° \leq a_{n,impar} \leq 90°$, por ejemplo, para la figura anterior, el ángulo 40° es equivalente a 140°. Esto es importante ya que determinará el signo de cada componente horizontal al momento de realizar los cálculos.
- Para las cuerdas agregadas con n par, el ángulo a_n debe de ser $90° \leq a_{n,par} \leq 0°$.

Nuestro objetivo es encontrar las tensiones T_1 y T_2 de dos cuerdas sosteniendo un peso $W = 1500\ N$, a un ángulo de $a_1 = 40°$ y $a_2 = 32°$ variando los valores de ángulo y tensión ya mencionados para cada cuerda agregada.

3.2. Expresión para T_1 y T_2

Tomando en cuenta los detalles anteriores, encontraremos la solución por sumatoria de fuerzas. Para el eje x se tiene

$$\begin{aligned}\sum F_x &= 0\\ 0 &= -T_1 \cos a_1 + T_2 \cos a_2 + S_x\\ T_1 &= \frac{1}{\cos a_1}(T_2 \cos a_2 + S_x)\end{aligned}$$

Notemos que tomamos el componente de T_1 como negativo lo más pronto posible por conveniencia, entonces los valores que puede tener a_1 son $90° \leq a_1 \leq 0°$.

Para el eje y se tiene

$$
\begin{aligned}
\sum F_y &= 0 \\
0 &= T_1 \sin a_1 + T_2 \sin a_2 + S_y - W \\
0 &= \left[\frac{1}{\cos a_1}(T_2 \cos a_2 + S_x)\right] \sin a_1 + T_2 \sin a_2 + S_y - W \\
0 &= T_2 \cdot \frac{\sin a_1 \cos a_2}{\cos a_1} + S_x \cdot \frac{\sin a_1}{\cos a_1} + T_2 \sin a_2 + S_y - W \\
0 &= T_2\left(\frac{\sin a_1 \cos a_2}{\cos a_1} + \sin a_2\right) + \frac{S_x \sin a_1}{\cos a_1} + S_y - W \\
0 &= T_2\left(\frac{\sin a_1 \cos a_2 + \sin a_2 \cos a_1}{\cos a_1}\right) + \frac{S_x \sin a_1}{\cos a_1} + S_y - W \\
0 &= T_2\left(\frac{\sin(a_1 + a_2)}{\cos a_1}\right) + \frac{S_x \sin a_1}{\cos a_1} + S_y - W \\
T_2\left(\frac{\sin(a_1 + a_2)}{\cos a_1}\right) &= -\frac{S_x \sin a_1}{\cos a_1} - S_y + W \\
T_2 \sin(a_1 + a_2) &= -S_x \sin a_1 - S_y \cos a_1 + W \cos a_1 \\
T_2 &= \frac{W \cos a_1 - (S_x \sin a_1 + S_y \cos a_1)}{\sin(a_1 + a_2)}
\end{aligned}
$$

Por lo tanto, las expresiones para T_1 y T_2 son:

$$
T_1 = \frac{1}{\cos a_1}(T_2 \cos a_2 + S_x),
$$

$$
T_2 = \frac{W \cos a_1 - (S_x \sin a_1 + S_y \cos a_1)}{\sin(a_1 + a_2)}.
$$

3.3. Implementación en Python 3

A continuación, se comparte el enlace directo al ejercicio en Google Colab:

https://colab.research.google.com/drive/1R5Ix2_4g2_-NeT9dW2GjJrg69ymomDHL?usp=sharing

- Creamos una clase para obtener las sumatorias S_x y S_y. Además agreguemos dos cuerdas, $T_3 = 50\ N$ con $a_3 = 120°$ y $T_4 = 70\ N$ con $a_4 = 40°$:

```
import math
conv = math.pi/180

class cuerda_extra:
  def __init__(self,t_n=0, a_n=0):
    self.t_n = t_n #tensi n variable
    self.a_n = a_n #angulo variable
  def compy(self):
    T_y=self.t_n * math.sin(self.a_n*conv) #componente en y
    return T_y
  def compx(self):
    T_x=self.t_n * math.cos(self.a_n*conv) #componente en x
    return T_x

#AGREGAR CUERDA EXTRA AQU
#tv_n=cuerda_extra(tensi n , angulo), para n impar 180<=a_n<=90 y para n par 90<=a_n
    <=0.
tensiones=[
cuerda_extra(50,120),
cuerda_extra(70,50)
]

print("Para las cuerdas agregadas se tiene:")
print("")
S_x = 0.0 #S_x es la sumatoria de componentes horizontales de las cuerdas.
for i in tensiones:
    S_x += i.compx()
print("Sumatoria S_x de fuerzas en eje x:",round(S_x,4), "newtons")

S_y = 0.0 #S_y es la sumatoria de componentes verticales de las cuerda.
for i in tensiones:
    S_y += i.compy()
print("Sumatoria S_y de fuerzas en eje y:",round(S_y,4), "newtons")
```

```
Para las cuerdas agregadas se tiene:

Sumatoria S_x de fuerzas en eje x: 19.9951 Newtons
Sumatoria S_y de fuerzas en eje y: 96.9244 Newtons
```

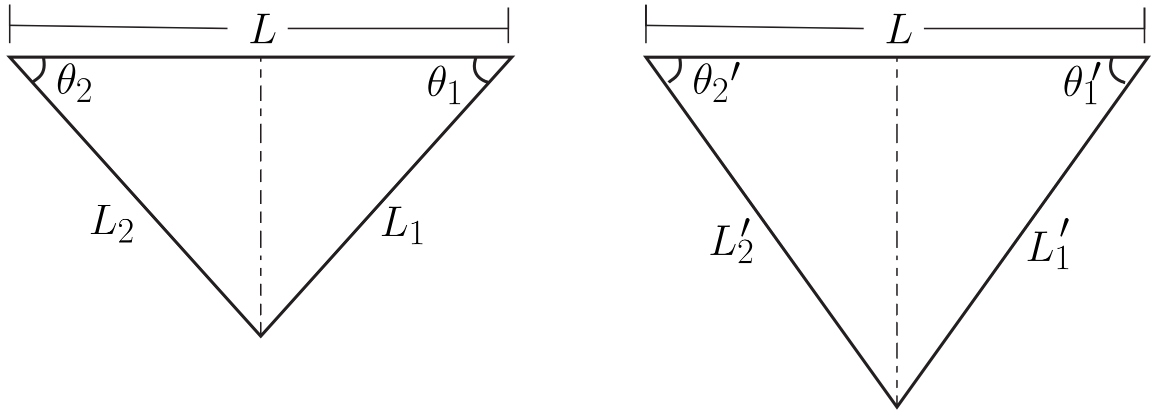
- Agregamos una clase para obtener el valor de las tensiones T_1 y T_2 utilizando las expresiones obtenidas anteriormente:

```
class Tensiones1y2:
  def __init__(self,a1=0,a2=0,W=0):
    self.a1 = a1 #angulo de tension T_1 con 90<=a1<=0
    self.a2 = a2 #angulo de tension T_2 con 90<=a2<=0
    self.W = W
  def calculoT_2(self):
    T2=(self.W*math.cos(self.a1*conv)-((S_x*math.sin(self.a1*conv))+(S_y*math.cos(self.
    a1*conv))))/(math.sin((self.a1+self.a2)*conv))
    return T2
  def calculoT_1(self):
    T1=(self.calculoT_2()*math.cos(self.a2*conv)+S_x)/(math.cos(self.a1*conv))
    return T1

t= Tensiones1y2(40,32,1500)

print("Valor de tension T_1:",round(t.calculoT_2(),2),"Newtons")
print("Valor de tension T_2:",round(t.calculoT_1(),2),"Newtons")
```

```
Valor de tension T_1: 1116.62 Newtons
Valor de tension T_2: 1262.25 Newtons
```

4. Problema 4

Este problema se reduce a un problema geométrico en el cual se debe calcular los ángulos finales dado los ángulos iniciales y sabiendo qué porcentaje se ha estirado la cuerda.



(a) Estado inicial sin elongación

(b) Estado final con elongación

Figura 2: Se muestran las dos configuraciones del sistema

Para empezar, asociamos al ángulo θ_1 a la longitud L_1 y al ángulo θ_2 a la longitud L_2, mientras que L será una longitud fija de la base superior. Las longitudes a estirar serán L_1 y L_2 de tal manera que

$$L_1 \to L_1', \qquad L_2 \to L_2'$$

Pero suponemos que se estiran un porcentaje e con $0 \le e \le 1$, entonces la nueva longitud será $L_1 = L_1 + eL_1$, así entonces tendremos

$$L_1' = (1+e)L_1, \qquad L_2' = (1+e)L_2 \tag{2}$$

Al hacer esto es evidente que los ángulos cambiarán y es de nuestro interés saber cuál será el valor de estos nuevos ángulos finales

$$\theta_1 \to \theta_1', \qquad \theta_2 \to \theta_2'$$

Con esto dicho, procedemos a calcular el valor de estos ángulos y lo hacemos empleando la ley de cosenos.

$$L_1'^2 = L^2 + L_2'^2 - 2LL_2' \cos\theta_2'$$

y usando las relaciones de (2)

$$L_1^2(1+e)^2 = L^2 + L_2^2(1+e)^2 - 2(1+e)LL_2 \cos\theta_2'$$

entonces

$$\cos\theta_2' = \frac{L^2 + L_2^2(1+e)^2 - L_1^2(1+e)^2}{2(1+e)LL_2} \cdot \frac{1/L_2^2}{1/L_2^2}$$

$$\cos\theta_2' = \frac{\left(\frac{L}{L_2}\right)^2 + (1+e)^2\left[1 - \left(\frac{L_1}{L_2}\right)^2\right]}{2(1+e)\left(\frac{L}{L_2}\right)} \tag{3}$$

pero del triángulo inicial tenemos las siguientes relaciones

$$L_2 \sin\theta_2 = L_1 \sin\theta_1, \qquad\qquad L = L_1 \cos\theta_1 + L_2 \cos\theta_2$$

los que también se puede escribir como

$$\frac{L_1}{L_2} = \frac{\sin\theta_2}{\sin\theta_1} = \sin\theta_2 \csc\theta_1 \tag{4}$$

y

$$\frac{L}{L_2} = \frac{L_1}{L_2}\cos\theta_1 + \cos\theta_2 = \sin\theta_2 \cot\theta_1 + \cos\theta_2 \tag{5}$$

sustituyendo (4) y (5) en (3) obtenemos

$$\theta_2' = \arccos\left[\frac{[\sin\theta_2\cot\theta_1 + \cos\theta_2]^2 + (1+e)^2\left[1 - \sin^2\theta_2\csc^2\theta_1\right]}{2(1+e)\left[\sin\theta_2\cot\theta_1 + \cos\theta_2\right]}\right]$$

si

$$\kappa_1 = \sin\theta_2\cot\theta_1 + \cos\theta_2, \qquad\qquad \kappa_2 = 1 - \sin^2\theta_2\csc^2\theta_1 \tag{6}$$

entonces

$$\theta_2' = \arccos\left[\frac{\kappa_1^2 + (1+e)^2\kappa_2}{2(1+e)\kappa_1}\right] \tag{7}$$

y por simetría, para calcular θ_1', en las ecuaciones (6) solo realizamos las siguientes transformaciones

$$\theta_1 \to \theta_2, \qquad\qquad \theta_2 \to \theta_1$$

esto es útil pues al hacer el programa solo debemos podemos hacer una sola función empleando la fórmula (7) y solo darle distintos parámetros.

Únicamente queda implementar estas ecuaciones a código de la siguiente manera.

1. Iniciamos creando la clase y el constructor, pidiendo los ángulos iniciales como parámetros y también el porcentaje de elongación

```
import math
class sist_dinamico:
  def __init__(self, a1 = 0, a2 = 0, e = 0,  N=0):
    self.a1 = math.radians(a1)
    self.a2 = math.radians(a2)
    self.N = N
    self.e = e/100
    self.angulos_finales = dict()
    self.angulo_final()
```

   y en la última linea llamamos al método que calcula los ángulos finales para que este cálculo este listo cuando se realice la instancia.

2. Seguido de esto implementamos las fórmulas (6) y (7) en un método. Estas se aplicarán si los valores de e son del 0 %, 1.5 %, 10 % y 40 %.

```
def __calc_angulo_final(self, ang1, ang2, e):
    a_f = 0
    if e == 0.015 or e == 0.1 or e == 0.4 or e == 0:
      k1 = math.sin(ang1)/math.tan(ang2) + math.cos(ang1)
      k2 = 1 - math.pow(math.sin(ang1), 2)/math.pow(math.sin(ang2), 2)
      a_f = math.acos((math.pow(k1, 2) + math.pow((1+e), 2)*k2)/(2*(1+e)*k1))
      a_f = math.degrees(a_f)
    else:
      print("elongacion no valida, devolvemos valor cero")
    return a_f
```

3. Luego, creamos un nuevo método que calcule los dos ángulos finales empleando la misma fórmula para ambos ángulos definida en el método anterior y se guarda en un diccionario.

```
def angulo_final(self):
    self.angulos_finales["Angulo final 1"] = self.__calc_angulo_final(self.a1, self.a2,
     self.e)
    self.angulos_finales["Angulo final 2"] = self.__calc_angulo_final(self.a2, self.a1,
     self.e)
    return self.angulos_finales
```

4. por último creamos una instancia para probar el correcto funcionamiento

```
sist1 = sist_dinamico(a1 = 30.96,
                      a2 = 45,
                      e = 1.5)
```

lo cual nos da como resultado

```
sist1.angulos_finales
{'Angulo final 1': 31.79244950092161, 'Angulo final 2': 46.3987374824558}
```

5. Problema 5

El objetivo de este programa es crear n puntos aleatorios en el espacio $\mathbb{R}^3$, con $0 \leq n \leq 10$ que se denominarán posiciones iniciales. Para cada puntos se hará una instancia, los cuales tendrá atributos de posiciones iniciales y finales. Luego se deben crear N vectores aleatorios con $2 \leq N \leq 100$ para cada posición inicial y serán sumados entre sí y a la posición inicial respectiva. Resultando así en posiciones finales que se guardarán como atributo de cada objeto (cada punto).

Los vectores aleatorios deben ser de magnitud entre 0 y 10. Para lograr esto nos apoyamos en el uso de coordenadas esféricas. Son las siguientes

$$\begin{aligned} x &= r \sin\theta \cos\phi \\ y &= r \sin\theta \sin\phi \\ z &= r \cos\theta \end{aligned} \tag{8}$$

el siguiente gráfico ilustra el uso de estas coordenadas.

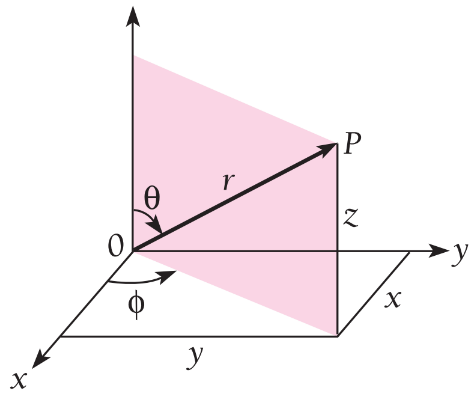


Figura 3: Coordenadas esféricas

Como sabemos que $x^2 + y^2 + z^2 = r^2$, entonces basta con establecer que $0 \leq r \leq 10$ para la condición de la magnitud de los vectores. Mientras que para los ángulos pueden variar como $0 \leq \theta \leq \pi$ y $0 \leq \phi \leq 2\pi$ aleatoriamente. De esta manera obtenemos coordenadas x, y, z aleatorias de magnitud acotada.

El código posee la siguiente estructura

1. Importamos librerias necesarias y creamos la clase para los puntos y establecemos atributos de posición inicial y final.

```
import random
import math
class p_aleatorios:
  def __init__(self):
    self.x_inicial = 0
    self.y_inicial = 0
    self.z_inicial = 0

    self.x_final = 0
    self.y_final = 0
    self.z_final = 0
```

2. Creamos un método que establece coordenadas iniciales a los puntos

```
def coor_inicial(self):
    self.x_inicial = random.randint(1,1000)
    self.y_inicial = random.randint(1,1000)
    self.z_inicial = random.randint(1,1000)
```

3. Creamos un método que crea una cantidad N de vectores aleatorios empleando las ecuaciones (8) y en las variables sum_x, sum_y, sum_z guarda la suma de todos los vectores hechos componente a componente. De tal forma que el vector (sum_x, sum_y, sum_z) será el vector final de la suma total de todos los vectores.

```
  def vec_aleatorios(self, N_vec):
    sum_x = 0
    sum_y = 0
    sum_z = 0
    for vec in range(N_vec):
        r = random.randint(0,10)
        theta = math.radians(random.randint(0, 180))
        phi = math.radians(random.randint(0, 360))
        x1 = r*math.sin(theta)*math.cos(phi)
        y1 = r*math.sin(theta)*math.sin(phi)
        z1 = r*math.cos(theta)
        sum_x += x1
        sum_y += y1
        sum_z += z1
    self.x_final = self.x_inicial + sum_x
    self.y_final = self.y_inicial + sum_y
    self.z_final = self.z_inicial + sum_z
```

por último, sumamos la posición inicial con la suma de todos los vectores anteriores y esto lo guardamos como la posición final (como atributo de los puntos)

4. Ahora bien, fuera de la clase, creamos una función para instanciar en una sola ejecución una cantidad n de puntos y los guardará en un lista.

```
  def crear_puntos(n: int):
  lista_puntos = []
  if 1 <= n <= 10:
    for i in range(n):
        p = p_aleatorios()
        lista_puntos.append(p)
  else:
    print("ERROR: ingrese un numero entre 1 y 10")
  return lista_puntos
```

de esta manera todos los puntos son instanciados con posiciones iniciales de valor cero.

5. Lo siguiente será darle coordenadas iniciales a los puntos ya instanciados a través del uso de polimorfismo llamando al método coor_inicial() para cada objeto en la lista de objetos (puntos).

```
def establecer_coor_ini(objetos):
  for objeto in objetos:
    objeto.coor_inicial()
```

6. Luego creamos una función para hacer la suma de vectores a todos los puntos y así establecer las posiciones finales dada una cantidad N de vectores a sumar. Esto hace el llamado al método vec_aleatorios().

```
def sumar_vectores(objetos, N):
  if 2 <= N <= 100:
    for objeto in objetos:
      objeto.vec_aleatorios(N)
  else:
    print("No puede ingresar mas de 100 vectores ni menos que 2")
```

7. Por último, creamos una función para únicamente comparar los estados inicial y final de todos los puntos.

```
def ver_cambios(objetos):
   for objeto in objetos:
    print("Inicial: ", [objeto.x_inicial, objeto.y_inicial, objeto.z_inicial])
    print("Final:   ", [round(objeto.x_final, 2), round(objeto.y_final, 2), round(
    objeto.z_final, 2)])
```

Una vez hecho todo esto, procedemos a hacer pruebas del funcionamiento del programa. Por fines prácticos creamos solo 2 puntos y le sumamos 20 vectores.

- Instanciamos todos los puntos

```
puntos = crear_puntos(2)
puntos
```

  y obtenemos la lista de objetos (puntos), cada uno mostrado con su dirección en memoria.

```
[<__main__.p_aleatorios at 0x7d981e331270>,
 <__main__.p_aleatorios at 0x7d981e331030>]
```

- Si probamos la función `ver_cambios(puntos)`

```
Inicial:  [0, 0, 0]
Final:    [0, 0, 0]
Inicial:  [0, 0, 0]
Final:    [0, 0, 0]
```

  obtenemos posiciones iniciales y finales cero como era de esperarse.

- Establecemos las posiciones iniciales aleatorias

```
establecer_coor_ini(puntos)
```

  y si probamos una vez más la función `ver_cambios(puntos)` notamos que se han establecido correctamente las posiciones iniciales

```
Inicial:  [254, 423, 355]
Final:    [0, 0, 0]
Inicial:  [959, 361, 884]
Final:    [0, 0, 0]
```

- Por último, sumamos 20 vectores aleatorios a cada punto

```
sumar_vectores(puntos, 20)
```

  y al revisar los cambios obtenemos lo siguiente

```
Inicial:  [254, 423, 355]
Final:    [234.8, 404.7, 357.42]
Inicial:  [959, 361, 884]
Final:    [973.06, 366.05, 846.31]
```

  Justo como se esperaba, las posiciones finales han sido modificadas.