UNIVERSIDAD NACIONAL SAN AGUSTIN DE AREQUIPA

Facultad de Ingenierías de Produccion y Servicios

Laboratorio

Fisica Computacional

Nombre Apellidos:

Fuentes Paredes Nelson Alejandro

Mayo 2020

[1]:
```
%matplotlib inline
#%matplotlib notebook

import sys
sys.path.append("../") # go to parent dir
```

1 Importando Modelos

[2]:
```
from models.Vector import Vector, Coordenate
from models.Graphic import Graphic
```

Codigo de los modelos en los Anexos

2 Importando Librerias

[3]:
```
import numpy
import math
```

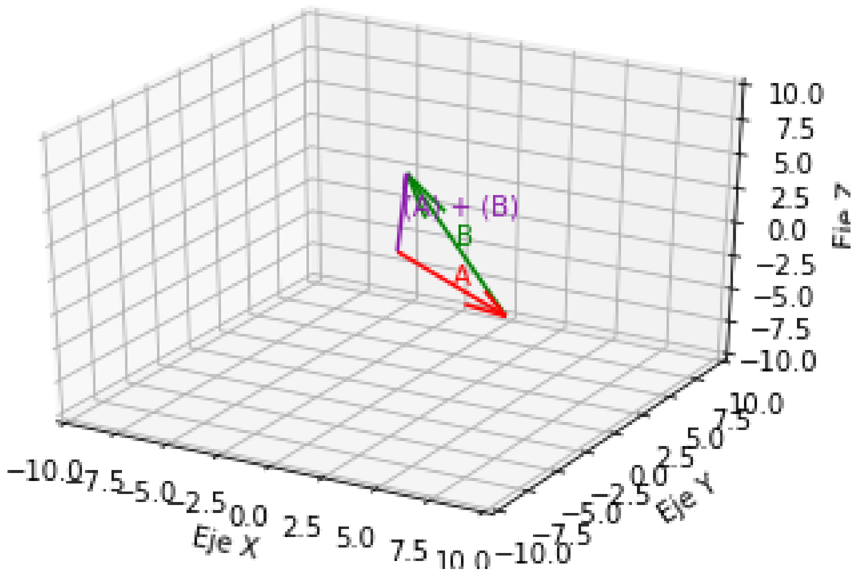
3 Ejercicio 1

Sean los vectores A~ = 3i + 4j − 6k y B~ = −6i + 2j + 8k, encuentre gr´aficamente. Verifique que los resultados estan en un plano.

[4]:
```
A = Vector( value=Coordenate(3, 4, -6), label="A", color='r')
B = Vector(origin=A.destiny,value=Coordenate(-6,2,8), label="B", color='g')
```

(a) R = A + B , |R|, sus cosenos directores cos 1 = Rx/R, cos 2 = Ry/R, cos 3 = Rz/R

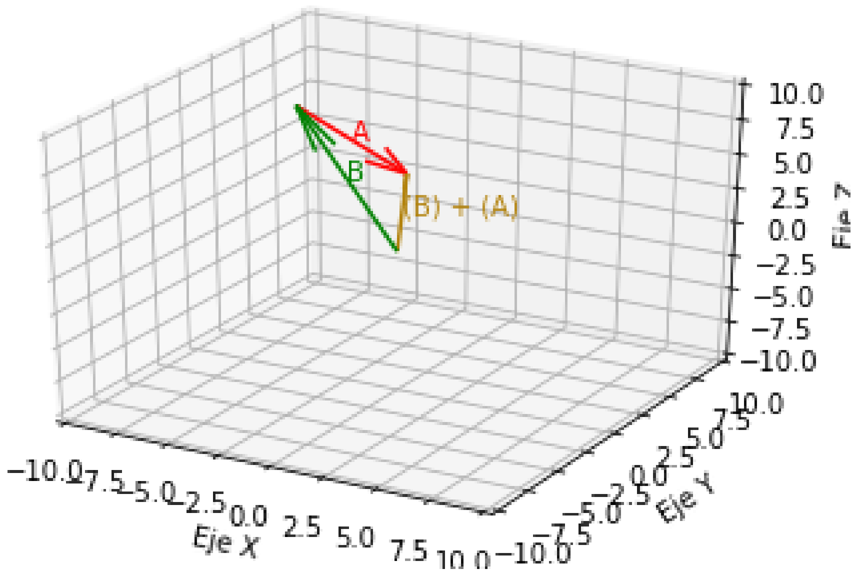
[5]:
```
R = A+B
G = Graphic()
G.vector(A)
G.vector(B)
G.vector(R)
G.show()
print('A:{} + B:{} = R:{}'.format(A,B,R))
print('Longitud R = {}'.format(R.length))
print('Cos(a1) = {}'.format(R.value.x/R.length))
print('Cos(a2)= {}'.format(R.value.y/R.length))
print('Cos(a3) = {}'.format(R.value.z/R.length))
```




```
A:[3.00; 4.00; -6.00] + B:[-6.00; 2.00; 8.00] = R:[-3.00; 6.00; 2.00]
Longitud R = 7.0
Cos(a1) = -0.42857142857142855
Cos(a2)= 0.8571428571428571
Cos(a3) = 0.2857142857142857
```

(b) R = B + A

[6]:
```
B.setOrigin()
A.setOrigin(B.destiny)
R = B+A
G = Graphic()
G.vector(A)
G.vector(B)
G.vector(R)
G.show()
print('A:{} + B:{} = R:{}'.format(A,B,R))
print('Longitud R = {}'.format(R.length))
print('Cos(a1) = {}'.format(R.value.x/R.length))
print('Cos(a2) = {}'.format(R.value.y/R.length))
print('Cos(a3) = {}'.format(R.value.z/R.length))
```

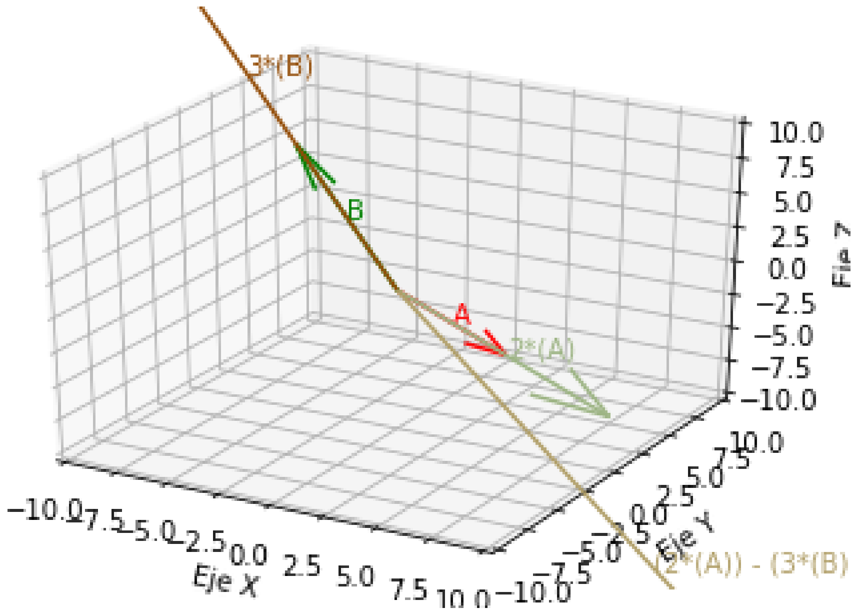



```
A:[3.00; 4.00; -6.00] + B:[-6.00; 2.00; 8.00] = R:[-3.00; 6.00; 2.00]
Longitud R = 7.0
Cos(a1) = -0.42857142857142855
Cos(a2) = 0.8571428571428571
Cos(a3) = 0.2857142857142857
```

(c) S = 2A − 3B

[7]:

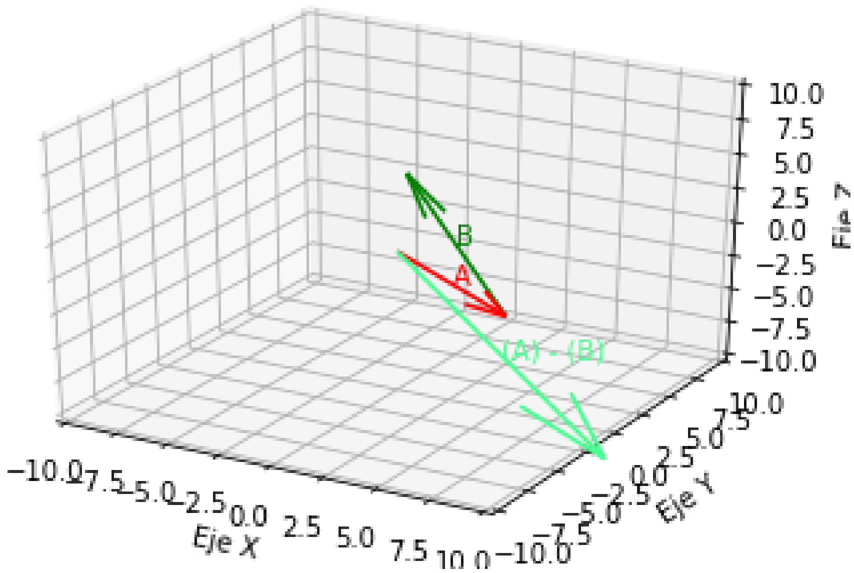
```
A.setOrigin()
A_2 = A.mulEscalar(2)
B_3 = B.mulEscalar(3)
#B_3.setOrigin(A_2.destiny)
S = A_2-B_3
G = Graphic()
G.vector(A)
G.vector(A_2)
G.vector(B)
G.vector(B_3)
G.vector(S)
G.show()
print('2*A:{} - 3*B:{} = S:{}'.format(A_2,B_3,S))
print('Longitud S = {}'.format(S.length))
print('Cos(a1) = {}'.format(S.value.x/S.length))
print('Cos(a2) = {}'.format(S.value.y/S.length))
print('Cos(a3) = {}'.format(S.value.z/S.length))
```




```
2*A:[6.00; 8.00; -12.00] - 3*B:[-18.00; 6.00; 24.00] = S:[24.00; 2.00; -36.00]
Longitud S = 43.31281565541543
Cos(a1) = 0.5541085158475322
Cos(a2) = 0.046175709653961 02
Cos(a3) = -0.8311627737712983
```

(d) D = A − B

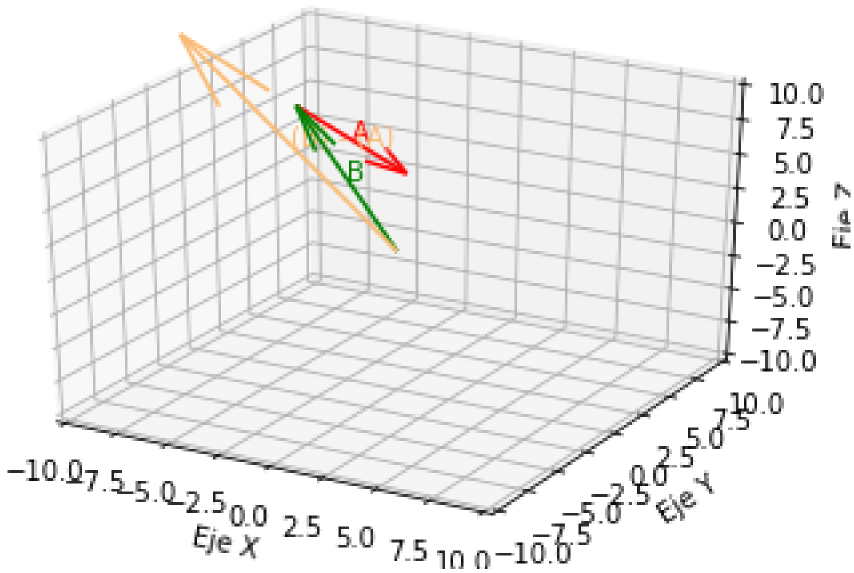
[8]:
```python
A.setOrigin()
B.setOrigin(A.destiny)
D = A-B
G = Graphic()
G.vector(A)
G.vector(B)
G.vector(D)
G.show()
print('A:{} - B:{} = D:{}'.format(A,B,D))
print('Longitud D = {}'.format(D.length))
print('Cos(a1) = {}'.format(D.value.x/D.length))
print('Cos(a2) = {}'.format(D.value.y/D.length))
print('Cos(a3) = {}'.format(D.value.z/D.length))
```




```
A:[3.00; 4.00; -6.00] - B:[-6.00; 2.00; 8.00] = D:[9.00; 2.00; -14.00]
Longitud D = 16.76305461424021
Cos(a1) = 0.5368949876447042
Cos(a2) = 0.11930999725437871
Cos(a3) = -0.835169980780651
```

(e) D~ = B~ − A~

[9]:
```
B.setOrigin()
A.setOrigin(B.destiny)
D =B-A
G = Graphic()
G.vector(A)
G.vector(B)
G.vector(D)
G.show()
print('B:{} - A:{} = D:{}'.format(B,A,D))
print('Longitud D = {}'.format(D.length))
print('Cos(a1) = {}'.format(D.value.x/D.length))
print('Cos(a2) = {}'.format(D.value.y/D.length))
print('Cos(a3) = {}'.format(D.value.z/D.length))
```




```
B:[-6.00; 2.00; 8.00] - A:[3.00; 4.00; -6.00] = D:[-9.00; -2.00; 14.00]
Longitud D = 16.76305461424021
Cos(a1) = -0.5368949876447042
Cos(a2) = -0.11930999725437871
Cos(a3) = 0.835169980780651
```

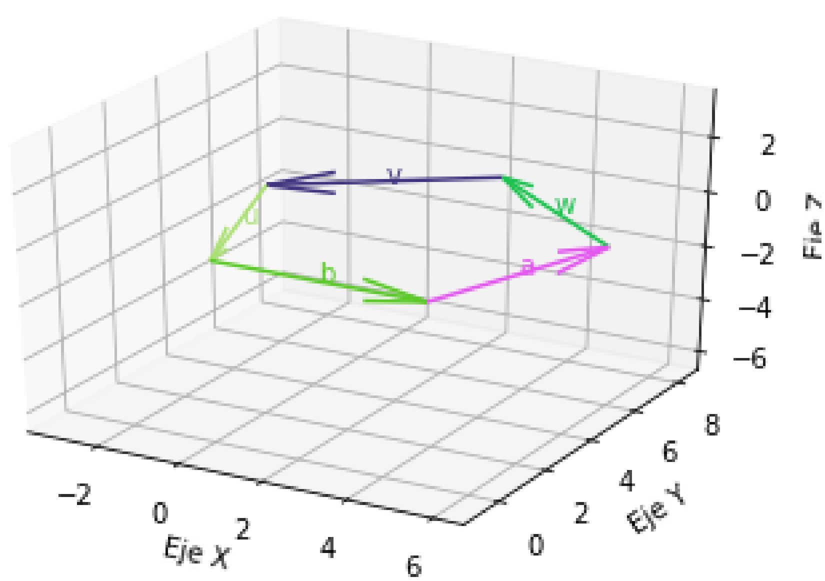
4 Ejercicio 2

Dibuje un pentágono de lado 5 cm vectorialmente, hacer que cada lado del pentágono sea un vector. Demuestre que la suma de dichos vectores es igual a cero.

[10]:
```python
labels = ['u','v','w','a','b']

length = 5
u = Vector(value=Coordenate(math.cos(0)*length, math.cos(math.pi/2)*length, 0),␣
 ↪label=labels.pop(), color=numpy.random.rand(3,) )
vectors = [u]
prev = 0
ang_inter = 108 # Angulo interno de un pentagono regular
last = u

G = Graphic(xmin=-3.5, xmax=6.5, ymin=-1.25, ymax = 8.75, zmin = -6.5, zmax = 3.
 ↪5)
G.vector(u)

res = "{}: {}".format(last.label , last)
suma = last
```

```
while len(labels)!=0:
    ang_dec = prev + 180- ang_inter
    ang_rad = math.pi/180*ang_dec
    prev = ang_dec
    cos_ang = math.cos(ang_rad)
    cos_comp_ang = math.cos(math.pi/2-ang_rad)
    new_x = cos_ang*length
    new_y = cos_comp_ang*length
    new_z = 0
    last = Vector(origin = last.destiny ,value =Coordenate(new_x, new_y, new_z), label=labels.pop(), color=numpy.random.rand(3,))
    vectors.append(last)

    G.vector(last)
    res = "{} + {}: {}".format(res ,last.label , last)
    suma = suma + last
    print("{} = {}".format(res, suma) )


G.show()
```

```
b: [5.00; 0.00; 0.00] + a: [1.55; 4.76; 0.00] = [6.55; 4.76; 0.00]
b: [5.00; 0.00; 0.00] + a: [1.55; 4.76; 0.00] + w: [-4.05; 2.94; 0.00] = [2.50;
7.69; 0.00]
b: [5.00; 0.00; 0.00] + a: [1.55; 4.76; 0.00] + w: [-4.05; 2.94; 0.00] + v:
[-4.05; -2.94; 0.00] = [-1.55; 4.76; 0.00]
b: [5.00; 0.00; 0.00] + a: [1.55; 4.76; 0.00] + w: [-4.05; 2.94; 0.00] + v:
[-4.05; -2.94; 0.00] + u: [1.55; -4.76; 0.00] = [-0.00; 0.00; 0.00]
```

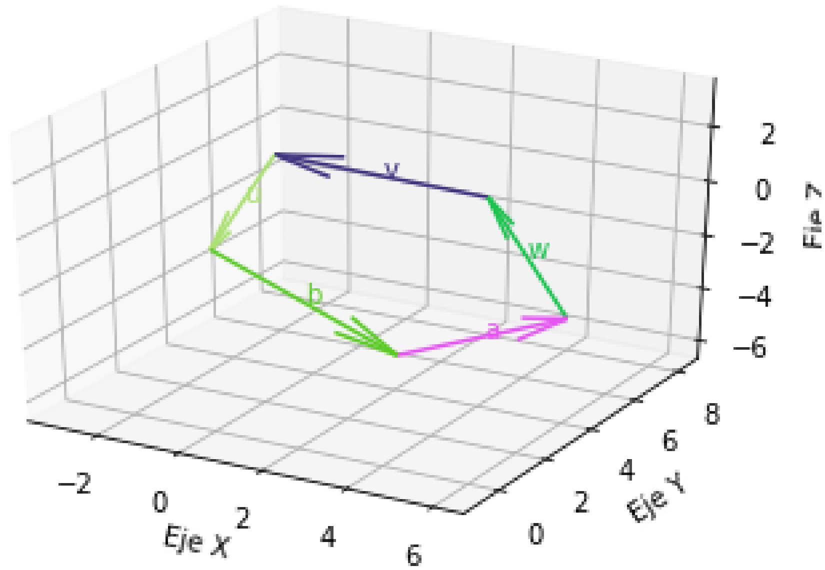


5 Ejercicio 3

Graficar dicho pent´agono de lado 5 cm vectorialmente en 3 dimensiones, sabiendo que dicho pent´agono esta perpendicular con el plano xz y forma un ´angulo de 30 ∘ con el eje x. Demuestre que la suma de dichos vectores es igual a cero

[11]:
```
G = Graphic(xmin=-3.5, xmax=6.5, ymin=-1.25, ymax = 8.75, zmin = -6.5, zmax = 3.
↪5)

res = "{}: {}".format(last.label , last)
suma = Vector()
origin = Coordenate(0,0,0)

#for vector in vectors:
    #G.vector(vector)
for vector in vectors:
    #G.vector(vector)
    print(vector.value)
    vector.rotateY(30)
    vector.setOrigin(origin)
    origin = vector.destiny
    G.vector(vector)
    res = "{} + {}: {}".format(res ,vector.label , vector)
    suma = suma + vector
    print("{} = {}".format(res, suma) )
G.show()
```

```
(5.0, 3.061616997868383e-16, 0)
u: [1.55; -4.76; 0.00] + b: [4.33; 0.00; -2.50] = [4.33; 0.00; -2.50]
(1.5450849718747373, 4.755282581475767, 0)
u: [1.55; -4.76; 0.00] + b: [4.33; 0.00; -2.50] + a: [1.34; 4.76; -0.77] =
[5.67; 4.76; -3.27]
(-4.045084971874736, 2.938926261462366, 0)
u: [1.55; -4.76; 0.00] + b: [4.33; 0.00; -2.50] + a: [1.34; 4.76; -0.77] + w:
[-3.50; 2.94; 2.02] = [2.17; 7.69; -1.25]
(-4.045084971874738, -2.938926261462365, 0)
u: [1.55; -4.76; 0.00] + b: [4.33; 0.00; -2.50] + a: [1.34; 4.76; -0.77] + w:
[-3.50; 2.94; 2.02] + v: [-3.50; -2.94; 2.02] = [-1.34; 4.76; 0.77]
(1.5450849718747361, -4.755282581475768, 0)
u: [1.55; -4.76; 0.00] + b: [4.33; 0.00; -2.50] + a: [1.34; 4.76; -0.77] + w:
[-3.50; 2.94; 2.02] + v: [-3.50; -2.94; 2.02] + u: [1.34; -4.76; -0.77] =
[-0.00; 0.00; 0.00]
```



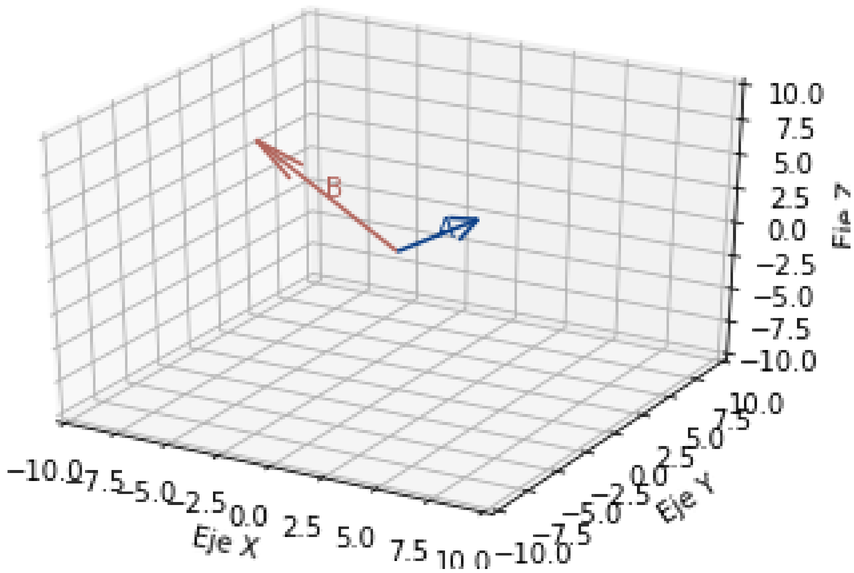
6 Ejercicio 4

Sean los vectores A = 6i − 4j + 6k, B = −8i + 2j + 5k y C = 2i − 7j + 3k. Determine y grafique según corresponda.

[12]:
```
A = Vector(value=Coordenate(6,-4,6), label='A', color=numpy.random.rand(3,))
B = Vector(value=Coordenate(-8,2,5), label='B', color=numpy.random.rand(3,))
C = Vector(value=Coordenate(2,-7,3), label='C', color=numpy.random.rand(3,))
```

(a) e = A · B y encuentre el ángulo entre estos vectores

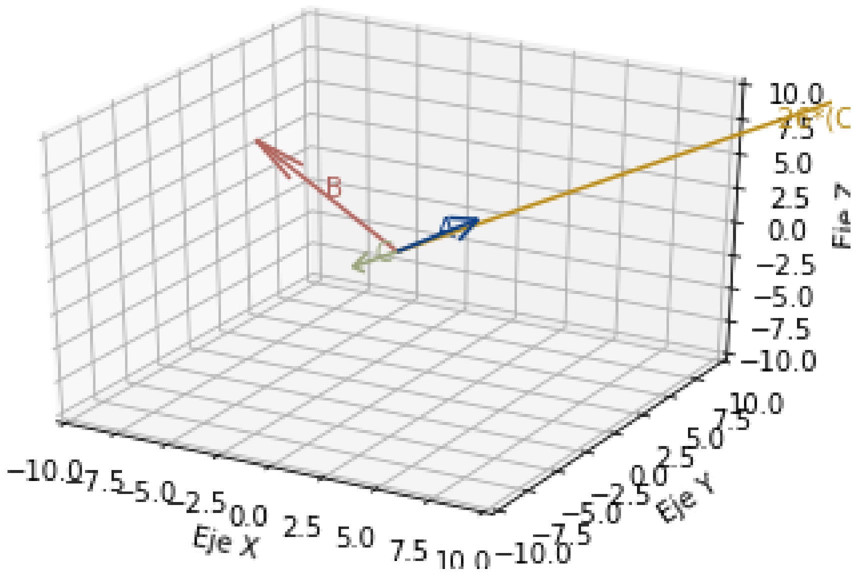
[13]:
```
G = Graphic()
G.vector(A)
G.vector(B)
G.show()
e = A * B
radiands = A.angle(B)
grades =  radiands*180/math.pi
print("Producto Punto e = A * B:" , e)
print("Angulo: {} rad, {}°".format(radiands, grades))
```




```
Producto Punto e = A * B: -26
Angulo: 1.86231013817353 rad, 106.70251106176846°
```

(b) $R = (A \cdot B)C$

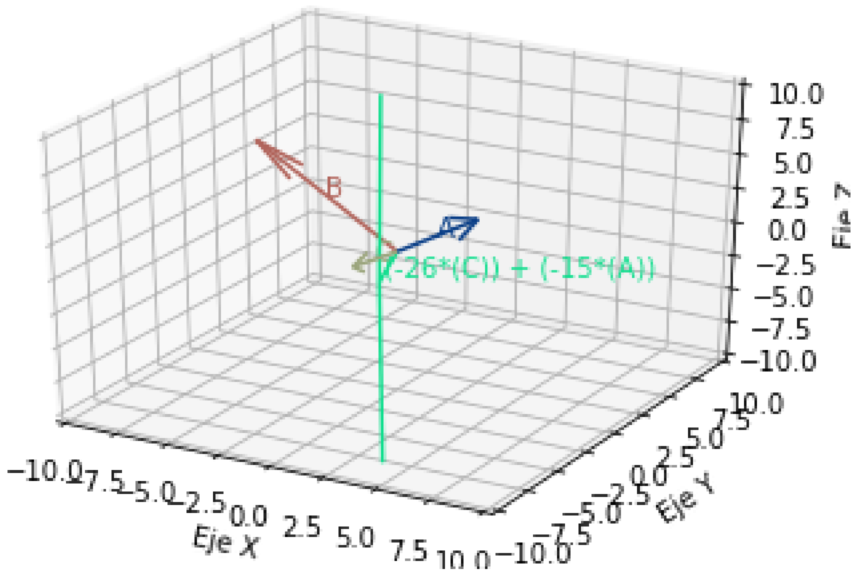
[14]:
```
R = C.mulEscalar(A * B)
G = Graphic()
G.vector(A)
G.vector(B)
G.vector(C)
G.vector(R)
G.show()
print("Producto Punto (A * B) C:" , R)
```




```
Producto Punto (A * B) C: [-52.00; 182.00; -78.00]
```

(c) $R = (A \cdot B)C + (B \cdot C)A$

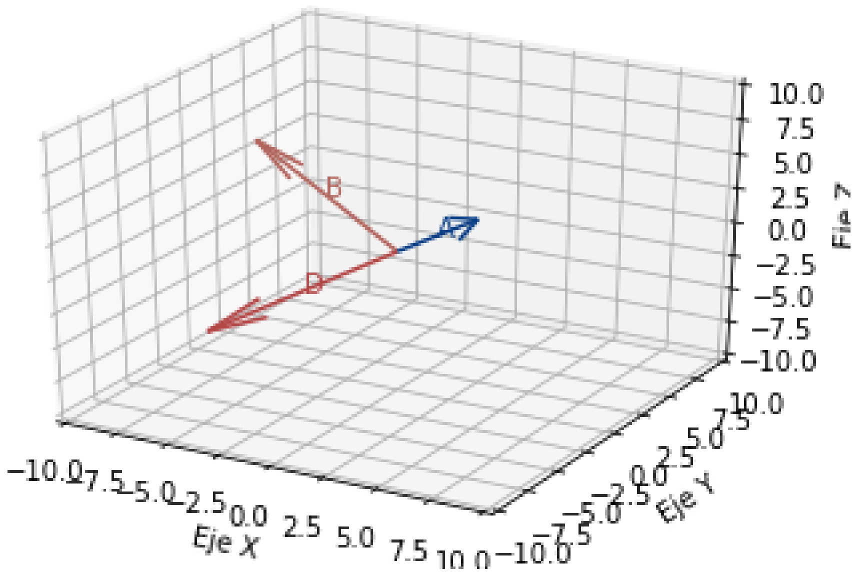
[15]:
```
R = C.mulEscalar(A * B)+ A.mulEscalar(B * C)
G = Graphic()
G.vector(A)
G.vector(B)
G.vector(C)
G.vector(R)
G.show()
print("Producto Punto (A * B) C + (B * C) A:" , R)
```




```
Producto Punto (A * B) C + (B * C) A: [-142.00; 242.00; -168.00]
```

(d) D = A X B

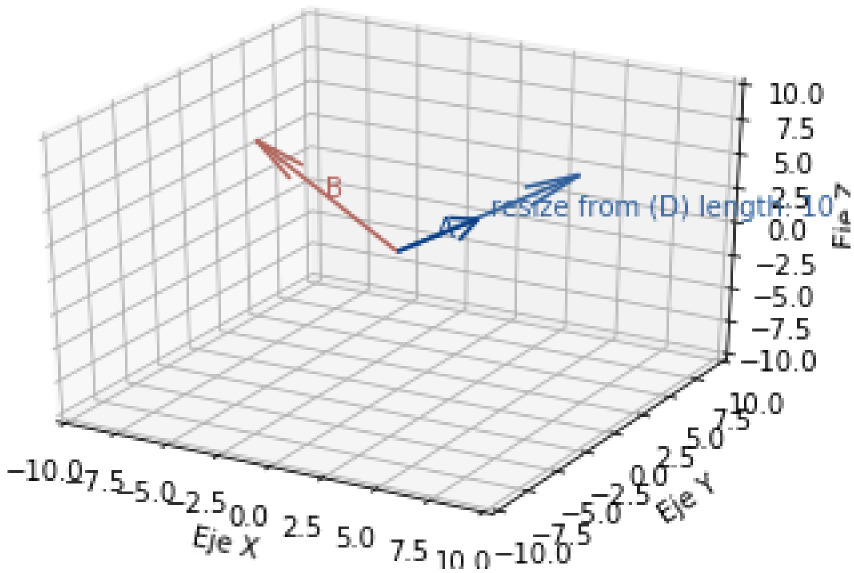
[16]:
```
D = A.productCrux(B, length=10)
D.setLabel('D')
G = Graphic()
G.vector(A)
G.vector(B)
G.vector(D)
G.show()
print("A X B = D:{}".format(D))
```




```
A X B = D:[-3.69; -9.00; -2.31]
```

(e) D = B x A

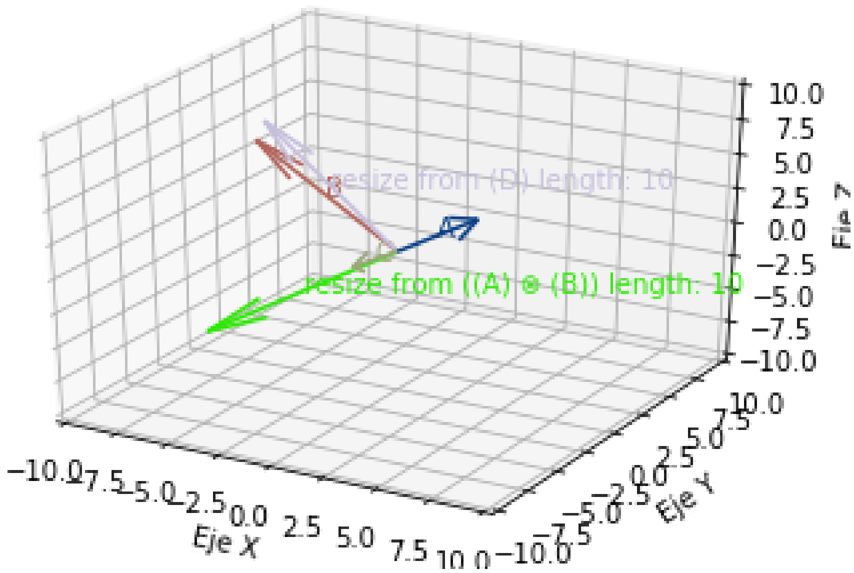
[17]:
```
D = B.productCrux(A)
D.setLabel('D')
G = Graphic()
G.vector(A)
G.vector(B)
G.vector(D.resize(10))
G.show()
print("B X A = D:{}".format(D))
```




```
B X A = D:[32.00; 78.00; 20.00]
```

(f) D = (A X B) X C

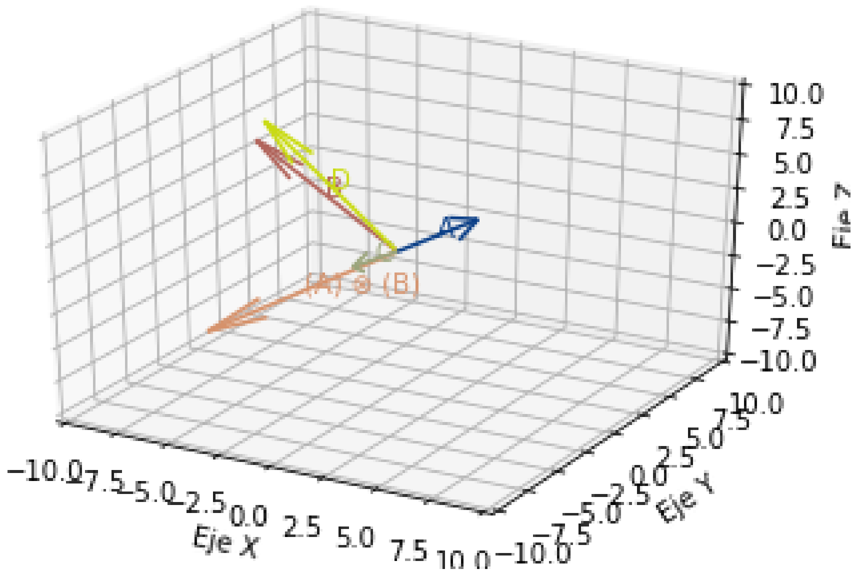
```
[18]: AXB = A.productCrux(B)
D = AXB.productCrux(C)
D.setLabel('D')
G = Graphic()
G.vector(A)
G.vector(B)
G.vector(C)
G.vector(D.resize(10))
G.vector(AXB.resize(10))
G.show()
print("(A X B) X C:{}".format(D))
```




```
(A X B) X C:[-374.00; 56.00; 380.00]
```

(g) D = A X (B X C)

```
[19]: AXB = A.productCrux(B, length=10)
      D = AXB.productCrux(C, length=10)
      D.setLabel('D')
      G = Graphic()
      G.vector(A)
      G.vector(B)
      G.vector(C)
      G.vector(D)
      G.vector(AXB)
      G.show()
      print("(A X B) X C:{}".format(D))
```




```
(A X B) X C:[-6.98; 1.04; 7.09]
```

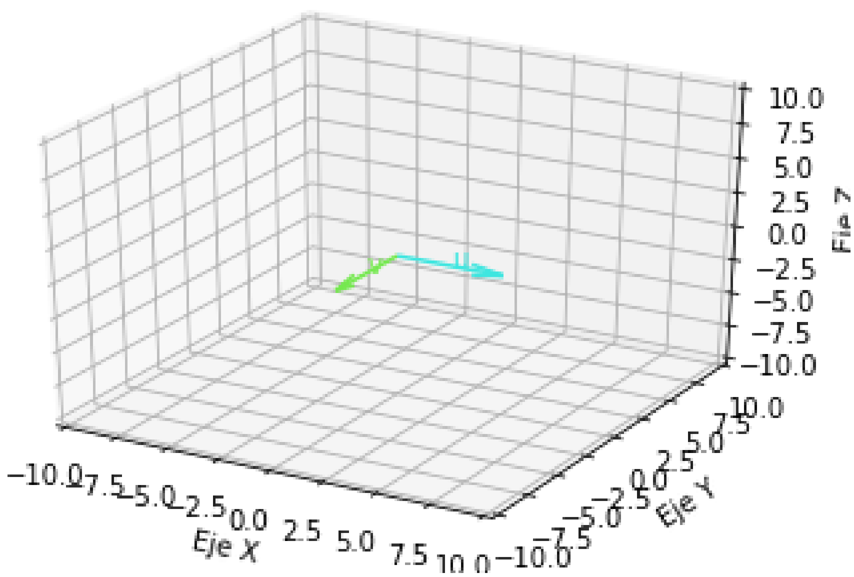
7 Ejercicio 5

Se ingresan dos vectores. Qué condición óptima aplica para que saber si los vectores son paralelos o perpendiculares?

[20]:
```
x1 = float(input("Ingrese coordenada X del Vector u:\t"))
y1 = float(input("Ingrese coordenada Y del Vector u:\t"))
z1 = float(input("Ingrese coordenada Z del Vector u:\t"))
u = Vector(
    value=Coordenate(
        x=x1,
        y=y1,
        z=z1
    ),
    label='u',
    color=numpy.random.rand(3, )
)

x2 = float(input("Ingrese dirección X del Vector v:\t"))
y2 = float(input("Ingrese dirección Y del Vector v:\t"))
z2 = float(input("Ingrese dirección Z del Vector v:\t"))
v = Vector(
    value=Coordenate(
        x=x2,
        y=y2,
```

```
        z=z2
    ),
    label='v',
    color=numpy.random.rand(3, )
)

G=Graphic()
G.vector(u)
G.vector(v)
G.show()
print("Son paralelos:\t{}\nSon perpendiculares:\t{}".format(u.areParallels(v),␣
 ↪u.arePerpendicular(v)))
```

```
Ingrese coordenada X del Vector u:      5
Ingrese coordenada Y del Vector u:      0
Ingrese coordenada Z del Vector u:      0
Ingrese dirección X del Vector v:       0
Ingrese dirección Y del Vector v:       -5
Ingrese dirección Z del Vector v:       0
```




```
Son paralelos:  False
Son perpendiculares:    True
```

8 Anexos

8.1 Clase Grafico

[2]:
```
import matplotlib.pyplot as plt
from Vector import Vector
import numpy as np
from mpl_toolkits.mplot3d import axes3d

class Graphic:

    DEFAULT_X_MIN = -10
    DEFAULT_X_MAX = 10
    DEFAULT_Y_MIN = -10
    DEFAULT_Y_MAX = 10
    DEFAULT_Z_MIN = -10
    DEFAULT_Z_MAX = 10


    def __init__(self, xmin=DEFAULT_X_MIN, xmax=DEFAULT_X_MAX,␣
↪ymin=DEFAULT_Y_MIN, ymax = DEFAULT_Y_MAX, zmin = DEFAULT_Z_MIN, zmax =␣
↪DEFAULT_Z_MAX):
        self.__figure__ = plt.figure()
        self.__axis__ = self.__figure__.gca(projection='3d')
        self.__axis__.set(xlim=(xmin, xmax), ylim=(ymin, ymax),␣
↪zlim=(zmin,zmax), xlabel='Eje X', ylabel='Eje Y', zlabel='Eje Z')

    def vector(self, v: Vector):
        self.__axis__.text((v.destiny.x + v.origin.x) / 2, (v.destiny.y + v.
↪origin.y) / 2, (v.destiny.z + v.origin.z) / 2, v.label, color=v.color)
        return  self.__axis__.quiver(v.origin.x, v.origin.y, v.origin.z, v.
↪value.x, v.value.y, v.value.z, color=v.color)

    def show(self):
        plt.show()
```

[ ]:

8.2 Clase Coordenada

[1]:
```
class Coordenate:
    DEFAULT_X = 0
    DEFAULT_Y = 0
    DEFAULT_Z = 0

    def __init__(self, x=DEFAULT_X, y=DEFAULT_Y, z=DEFAULT_X):
        self.__x__ = x
```

```
        self.__y__ = y
        self.__z__ = z

    @property
    def x(self):
        return self.__x__

    @property
    def y(self):
        return self.__y__

    def __add__(self, other):
        return Coordenate(x=self.__x__ + other.__x__, y=self.__y__ + other.
↪__y__, z=self.__z__ + other.__z__)

    def __sub__(self, other):
        return Coordenate(x=self.__x__ - other.__x__, y=self.__y__ - other.
↪__y__, z=self.__z__ - other.__z__)

    @property
    def z(self):
        return self.__z__

    def mulEscalar(self, number):
        return Coordenate(x=self.x * number, z=self.z * number, y=self.y *␣
↪number)

    def __str__(self):
        return '({}, {}, {})'.format(self.__x__, self.__y__, self.__z__)
```

8.3 Clase Vector

[2]:
```
class Vector:
    DEFAULT_ORIGIN = Coordenate()
    DEFAULT_VALUE = Coordenate()
    DEFAULT_LABEL = "v"
    DEFAULT_COLOR = "b"

    def __init__(self, origin=DEFAULT_ORIGIN, value=DEFAULT_VALUE,␣
↪label=DEFAULT_LABEL, color=DEFAULT_COLOR):
        self.__origin__ = origin
        self.__value__ = value
        self.__label__ = label
        self.__color__ = color

    def __add__(self, other):
```

```
        return Vector(origin=self.origin, value=self.value + other.value,␣
↪color=numpy.random.rand(3, ),
                      label='({}) + ({})'.format(self.label, other.label))

    def __sub__(self, other):
        return Vector(origin=self.origin, value=self.value - other.value,␣
↪color=numpy.random.rand(3, ),
                      label='({}) - ({})'.format(self.label, other.label))

    def setOrigin(self, origen=Coordenate()):
        self.__origin__ = origen

    def setLabel(self, label):
        self.__label__ = label

    @property
    def color(self):
        return self.__color__

    @property
    def label(self):
        return self.__label__

    @property
    def origin(self):
        return self.__origin__

    @property
    def destiny(self):
        return self.origin + self.value

    @property
    def value(self):
        return self.__value__

    @property
    def length(self):
        return float(math.sqrt(self.value.x ** 2 + self.value.y ** 2 + self.
↪value.z ** 2))

    def mulEscalar(self, number):
        return Vector(origin=self.origin, value=self.value.mulEscalar(number),
                      label=str(number) + '*(' + self.label + ')', color=numpy.
↪random.rand(3, ))

    def __str__(self):
```

```
        return "[{}; {}; {}]".format("{:.2f}".format(self.value.x), "{:.2f}".
↪format(self.value.y),
                                    "{:.2f}".format(self.value.z))

    def __matrixRotationZ(self, radians):
        return [
            [math.cos(radians), -1 * math.sin(radians), 0],
            [math.sin(radians), math.cos(radians), 0],
            [0, 0, 1]
        ]

    def __matrixRotationY(self, radians):
        return [
            [math.cos(radians), 0, math.sin(radians)],
            [0, 1, 0],
            [-1 * math.sin(radians), 0, math.cos(radians)]]

    def __matrixRotationX(self, radians):
        return [
            [1, 0, 0],
            [0, math.cos(radians), -1 * math.sin(radians)],
            [0, math.sin(radians), math.cos(radians)]]

    def list(self):
        return [[ self.value.x ],[ self.value.y],[self.value.z ]]

    def rotateZ(self, grades):
        radians = grades*math.pi/180
        self.__rotate__(self.__matrixRotationZ(radians))

    def rotateY(self, grades):
        radians = grades * math.pi / 180
        self.__rotate__(self.__matrixRotationY(radians))

    def rotateX(self, grades):
        radians = grades * math.pi / 180
        self.__rotate__(self.__matrixRotationX(radians))

    def __rotate__(self, matrix):
        array = numpy.dot(matrix, self.list())
        x, y, z = numpy.transpose(array).tolist()[0]
        length = self.length
        self.__value__ = Coordenate(x, y, z)

    def __mul__(self, vector):
        return self.value.x * vector.value.x + self.value.y * vector.value.y +␣
↪self.value.z * vector.value.z
```

```
    def angle(self, vector):
        return float(math.acos((self * vector)/(self.length * vector.length)))

    def productCrux(self, vector, length = None):
        label = "({})   ({})".format(self.label, vector.label)
        crux = Vector(
            value=Coordenate(
                x=self.value.y * vector.value.z - self.value.z * vector.value.y,
                y=self.value.z * vector.value.x - self.value.x * vector.value.z,
                z=self.value.x * vector.value.y - self.value.y * vector.value.
↪x),
            label=label,
            color=numpy.random.rand(3, )
        )
        if length is not None:
            crux_resize = crux.resize(length)
            crux_resize.setLabel(label)
            return crux_resize
        else:
            return crux

    def unitaryVector(self):
        length = self.length
        return Vector(
            value=Coordenate(
                x=self.value.x/length,
                y=self.value.y/length,
                z=self.value.z/length,
            ),
            label="unitary({})".format(self.label),
            color=numpy.random.rand(3, )
        )

    def resize(self, length):
        new_vector = self.unitaryVector()
        new_vector.__value__ = Coordenate(
            x=new_vector.value.x*length,
            y=new_vector.value.y*length,
            z=new_vector.value.z*length
        )
        new_vector.setLabel("resize from ({}) length: {}".format(self.label,␣
↪length))
        return new_vector

    def areParallels(self, vector):
        try:
```

```
            dx = self.value.x  / vector.value.x
            dy = self.value.y / vector.value.y
            dz = self.value.z / vector.value.z
            return dx == dy and dx == dz
        except:
            return False

    def arePerpendicular(self, vector):
        return self * vector == 0
```

[ ]:

```

```