Universidad Nacional San Agustin de Arequipa

Facultad de Ingenierias de Produccion y Servicios

Escuela Profesional de Ingenieria de Sistemas

Fisica Computacional

Alumno:

Fuentes Paredes Nelson Alejandro

Mayo 2020

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

1 Importando Librerias

[2]:
```
import matplotlib.pyplot as plt
import math
import numpy as np
```

2 Importando Modelos

[3]:
```
from models.Graphic import Graphic, Graphic3D
from models.Vector import Vector, Coordenate
```

Codigo de los modelos en los anexos

3 Metodos

3.1 Graficar Vector 2D

[4]:
```
def vector2D(axis, vector):
    axis.quiver(vector.origin.x, vector.origin.y,  vector.x,vector.y ,␣
 ↪angles='xy', scale_units='xy', scale=1)
```

3.2 Graficar Punto 2D

[5]:
```
def point2D(axis, coordenate):
    axis.plot(coordenate.x, coordenate.y, marker="o", )
```

4 a = 0, no hay aceleración

4.1 Velocidad constante en una dimensión

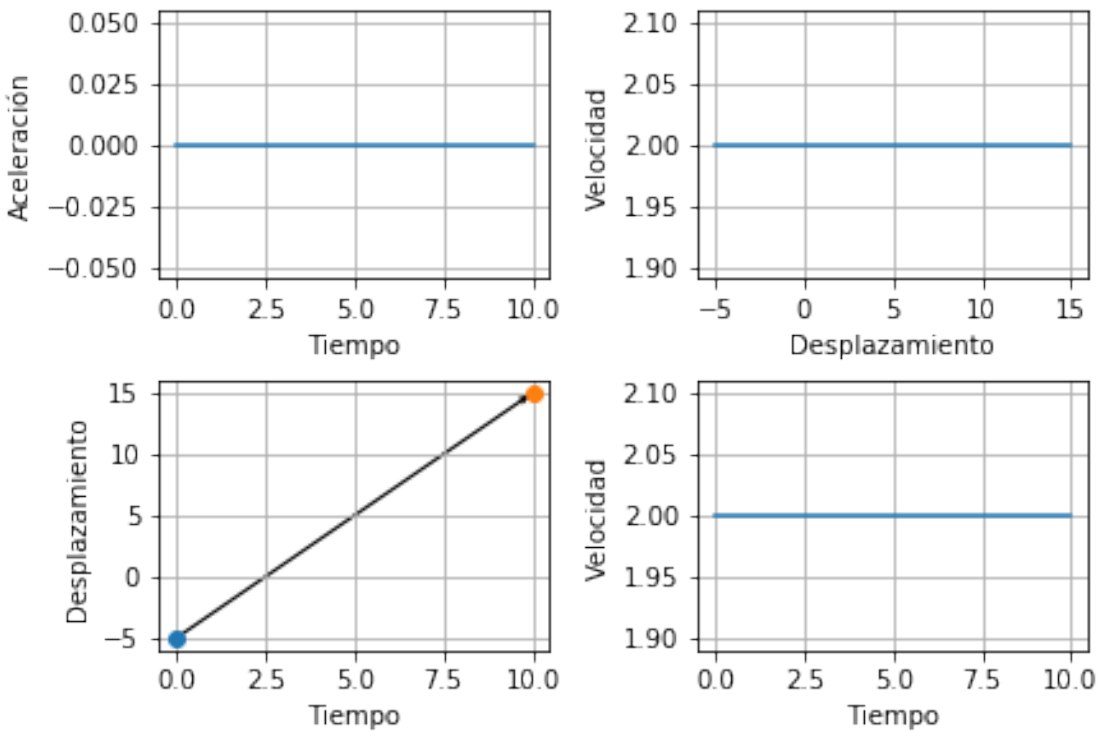
4.1.1 Ejericio 1

Sea una partícula, cuya posición inicial es $t_\circ = 0$, $y_\circ = -5$ m y $v = 2$ m/s. Determine la posición de la partícula para $t = 10$ s. Qué dirección toma?

[27]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema

t0 = 0                                          # tiempo inicial
y0 = Vector(value=Coordenate(x=0,y=-5))              # punto inicial
v0 = Vector(value=Coordenate(x=1,y=2), origin=y0)  # vector velocidad
tf = 10                                         # vector del punto inicial
```

```
pv = v0.mulEscalar(tf - t0)                    # vector velocidad por el tiempo␣
 ↪total
yf = y0 + pv                                 # vector resultante del desplazamiento
a0 = Vector()

#point2D(axis[1][0], Coordenate())
#vector2D(axis[1][0], y0)
#vector2D(axis[1][0], yf)

point2D(axis[1][0], y0.destiny)
point2D(axis[1][0], yf.destiny)
vector2D(axis[1][0], pv)
axis[1][0].set(ylabel = 'Desplazamiento', xlabel='Tiempo')
axis[1][0].grid()



axis[1][1].plot([t0,tf],[v0.y , v0.y])
axis[1][1].set(ylabel = 'Velocidad', xlabel='Tiempo')
axis[1][1].grid()

axis[0][0].plot([t0,tf], [a0.x, a0.x])
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')

axis[0][1].plot([y0.y, yf.y], [v0.y, v0.y])
axis[0][1].grid()
axis[0][1].set(ylabel = 'Velocidad', xlabel='Desplazamiento')


print("Resultado:\t{}".format(yf.y))
```

```
Resultado:      15
```



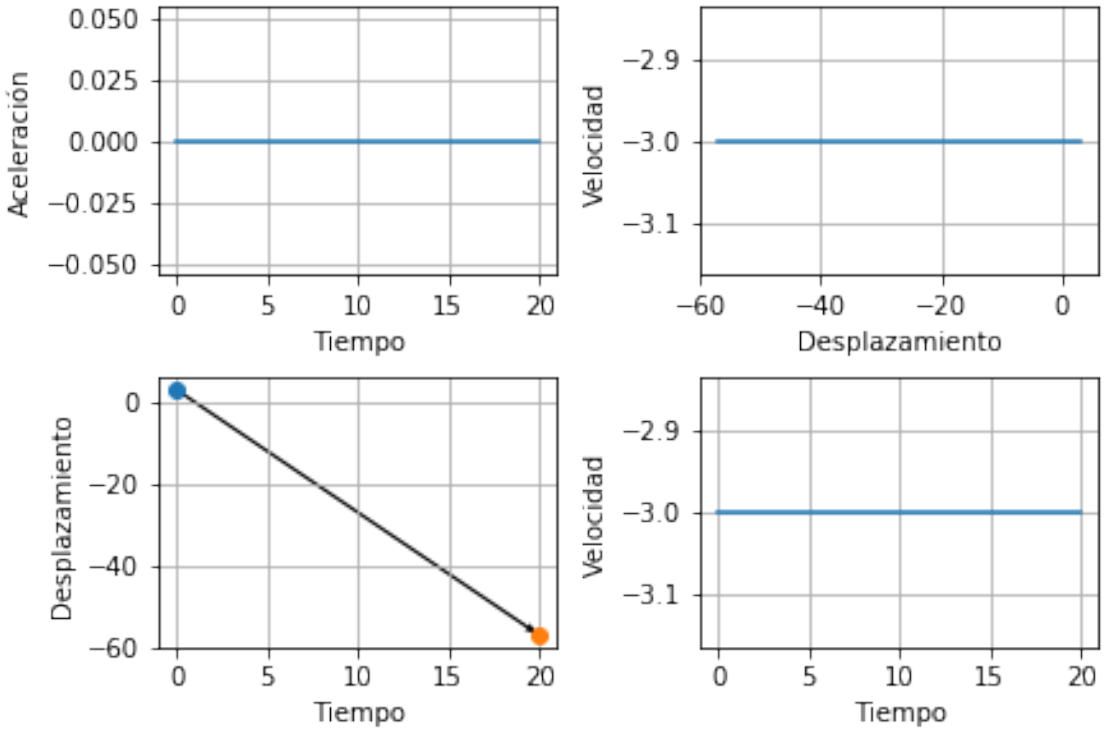
4.1.2 Ejercicio 2

Sea una particula, cuya posición inicial es t$_\circ$ = 0, z$_\circ$ = 3 m y vz = −3 m/s. Determine la posicion de la particula para t = 20 s. Que direccion toma?

[28]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema


t0 = 0                                          # tiempo inicial
y0 = Vector(value=Coordenate(x=0,y=3))               # punto inicial
v0 = Vector(value=Coordenate(x=1,y=-3), origin=y0)  # vector velocidad
tf = 20                                         # vector del punto inicial
pv = v0.mulEscalar(tf - t0)                    # vector velocidad por el tiempo␣
 ↪total
yf = y0 + pv                                # vector resultante del desplazamiento
a0 = Vector()

#point2D(axis[1][0], Coordenate())
#vector2D(axis[1][0], y0)
#vector2D(axis[1][0], yf)

point2D(axis[1][0], y0.destiny)
point2D(axis[1][0], yf.destiny)
```

```
vector2D(axis[1][0], pv)
axis[1][0].set(ylabel = 'Desplazamiento', xlabel='Tiempo')
axis[1][0].grid()


axis[1][1].plot([t0,tf],[v0.y , v0.y])
axis[1][1].set(ylabel = 'Velocidad', xlabel='Tiempo')
axis[1][1].grid()

axis[0][0].plot([t0,tf], [a0.x, a0.x])
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')

axis[0][1].plot([y0.y, yf.y], [v0.y, v0.y])
axis[0][1].grid()
axis[0][1].set(ylabel = 'Velocidad', xlabel='Desplazamiento')


print("Resultado:\t{}".format(yf.y))
```

```
Resultado:      -57
```

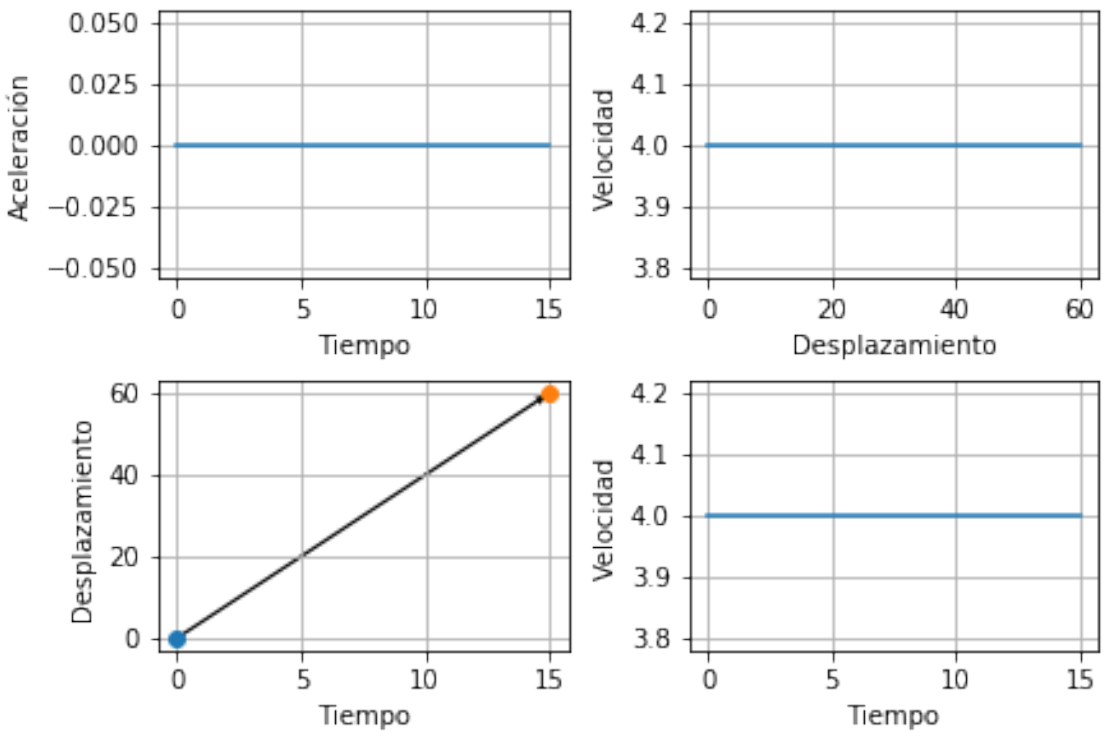


4.1.3 Ejercicio 3

Sea una partícula, cuya posición inicial es t$\circ$ = 0, x$\circ$ = 0 m y vx = 4 m/s. Determine la posición de la partícula para t = 15 s. Que dirección toma?

[29]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema


t0 = 0                                          # tiempo inicial
y0 = Vector(value=Coordenate(x=0,y=0))               # punto inicial
v0 = Vector(value=Coordenate(x=1,y=4), origin=y0)  # vector velocidad
tf = 15                                         # vector del punto inicial
pv = v0.mulEscalar(tf - t0)                    # vector velocidad por el tiempo␣
 ↪total
yf = y0 + pv                                 # vector resultante del desplazamiento
a0 = Vector()

#point2D(axis[1][0], Coordenate())
#vector2D(axis[1][0], y0)
#vector2D(axis[1][0], yf)

point2D(axis[1][0], y0.destiny)
point2D(axis[1][0], yf.destiny)
vector2D(axis[1][0], pv)
axis[1][0].set(ylabel = 'Desplazamiento', xlabel='Tiempo')
axis[1][0].grid()


axis[1][1].plot([t0,tf],[v0.y , v0.y])
axis[1][1].set(ylabel = 'Velocidad', xlabel='Tiempo')
axis[1][1].grid()

axis[0][0].plot([t0,tf], [a0.x, a0.x])
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')

axis[0][1].plot([y0.y, yf.y], [v0.y, v0.y])
axis[0][1].grid()
axis[0][1].set(ylabel = 'Velocidad', xlabel='Desplazamiento')


print("Resultado:\t{}".format(yf.y))
```

```
Resultado:      60
```



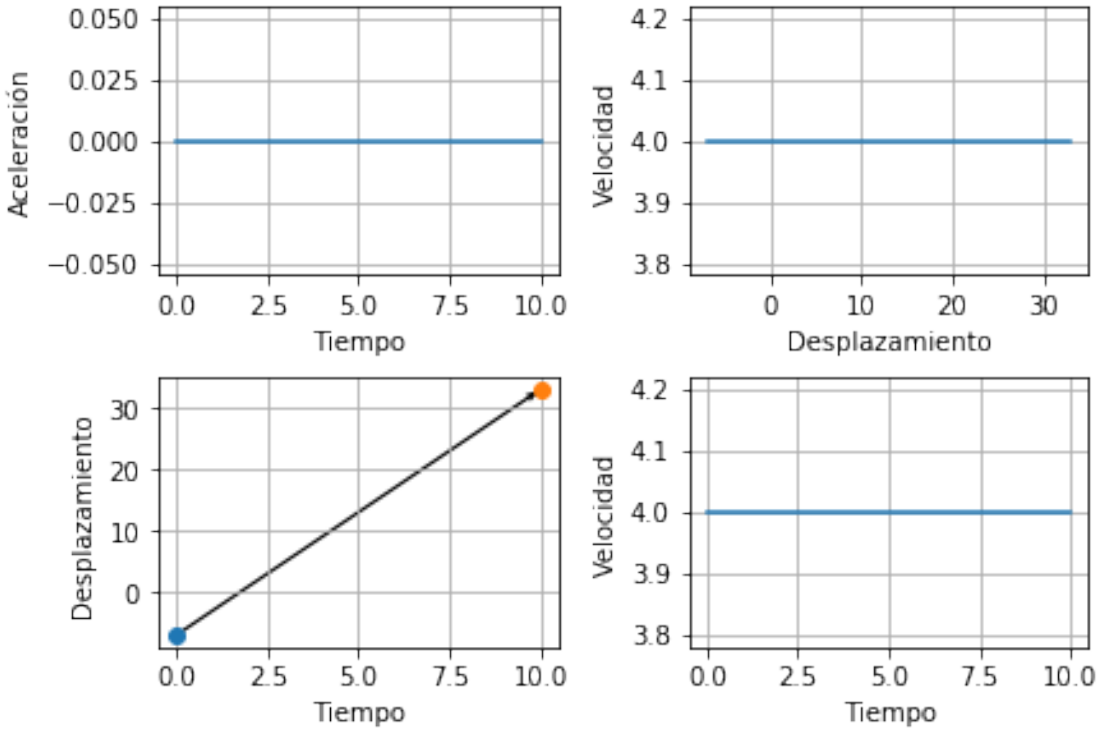
4.1.4 Ejercicio 4

Sea una partícula, cuya posición inicial es $t_\circ = 0$, $y_\circ = -7$ m y vy = 1 m/s. A su vez se manifiesta una velocidad del viento v = 3 m/s en la dirección y. Determine la posición de la partícula para t = 10 s. Qué dirección toma?

[30]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema

vp = 1                                          # velocidad de la particula
vw = 3                                          # velocidad del viento
t0 = 0                                           # tiempo inicial
y0 = Vector(value=Coordenate(x=0,y=-7))                 # punto inicial
v0 = Vector(value=Coordenate(x=1,y=vp+vw), origin=y0.destiny)  # vector␣
 ↪velocidad
tf = 10                                          # vector del punto inicial
pv = v0.mulEscalar(tf - t0)                     # vector velocidad de la␣
 ↪particula por el tiempo total
yf = y0 + pv                                # vector resultante del desplazamiento
a0 = Vector()

#point2D(axis[1][0], Coordenate())
#vector2D(axis[1][0], y0)
#vector2D(axis[1][0], yf)
```

```
point2D(axis[1][0], y0.destiny)
point2D(axis[1][0], yf.destiny)
vector2D(axis[1][0], pv)
axis[1][0].set(ylabel = 'Desplazamiento', xlabel='Tiempo')
axis[1][0].grid()


axis[1][1].plot([t0,tf],[v0.y , v0.y])
axis[1][1].set(ylabel = 'Velocidad', xlabel='Tiempo')
axis[1][1].grid()

axis[0][0].plot([t0,tf], [a0.x, a0.x])
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')

axis[0][1].plot([y0.y, yf.y], [v0.y, v0.y])
axis[0][1].grid()
axis[0][1].set(ylabel = 'Velocidad', xlabel='Desplazamiento')


print("Resultado:\t{}".format(yf.y))
```

```
Resultado:      33
```



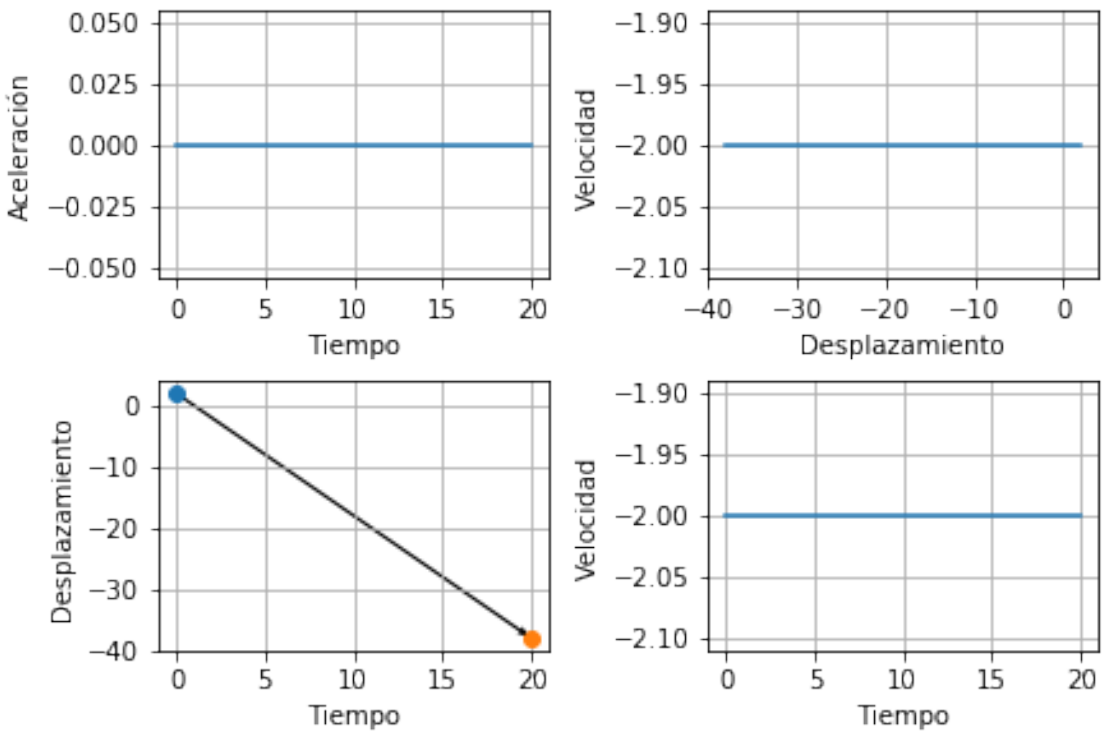
4.1.5 Ejercicio 5

Sea una partícula, cuya posición inicial es $t_\circ = 0$, $x_\circ = 2$ m y $vx = 1$ m/s. A su vez se manifiesta una velocidad del viento $v = -3$ m/s en la dirección x. Determine la posición de la partícula para $t = 20$ s. Qué dirección toma?

[26]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema

vp = 1                                          # velocidad de la particula
vw = -3                                          # velocidad del viento
t0 = 0                                           # tiempo inicial
y0 = Vector(value=Coordenate(x=0,y=2))                # punto inicial
v0 = Vector(value=Coordenate(x=1,y=vp+vw), origin=y0.destiny)  # vector␣
 ↪velocidad
tf = 20                                          # vector del punto inicial
pv = v0.mulEscalar(tf - t0)                     # vector velocidad de la␣
 ↪particula por el tiempo total
yf = y0 + pv                                 # vector resultante del desplazamiento
a0 = Vector()

#point2D(axis[1][0], Coordenate())
#vector2D(axis[1][0], y0)
#vector2D(axis[1][0], yf)


point2D(axis[1][0], y0.destiny)
point2D(axis[1][0], yf.destiny)
vector2D(axis[1][0], pv)
axis[1][0].set(ylabel = 'Desplazamiento', xlabel='Tiempo')
axis[1][0].grid()


axis[1][1].plot([t0,tf],[v0.y , v0.y])
axis[1][1].set(ylabel = 'Velocidad', xlabel='Tiempo')
axis[1][1].grid()

axis[0][0].plot([t0,tf], [a0.x, a0.x])
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')

axis[0][1].plot([y0.y, yf.y], [v0.y, v0.y])
axis[0][1].grid()
axis[0][1].set(ylabel = 'Velocidad', xlabel='Desplazamiento')
```

```
print("Resultado:\t{}".format(yf.y))
```

```
Resultado:      -38
```



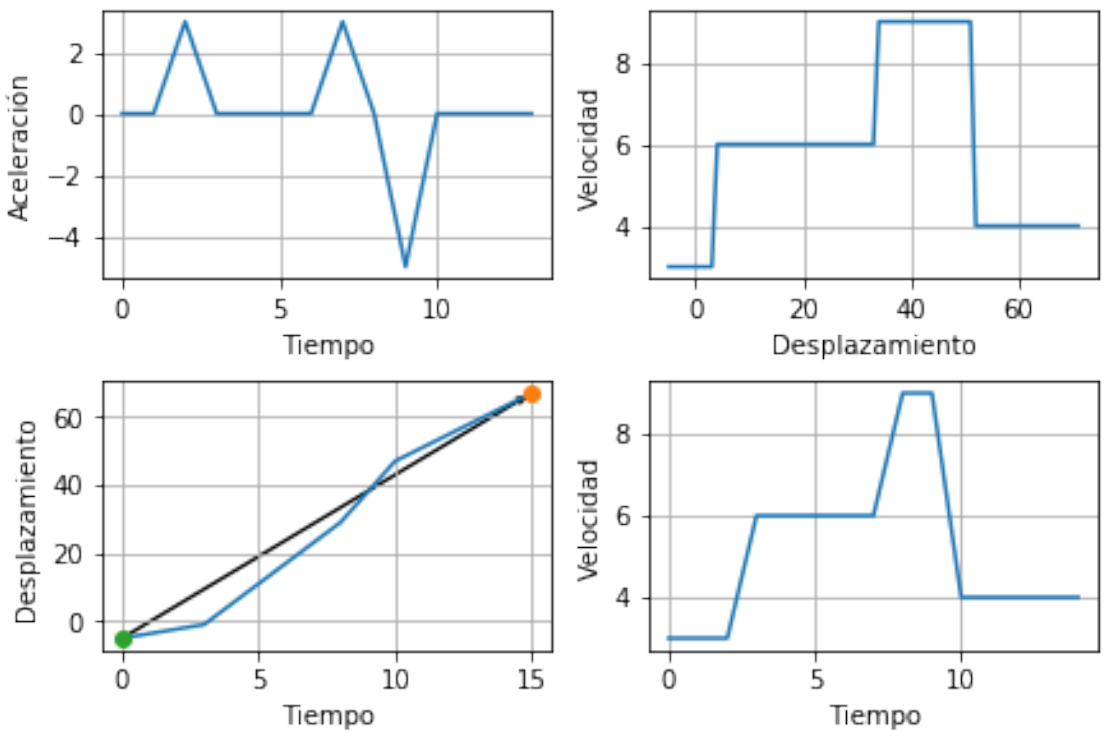
4.1.6 Ejercicio 6

La posición inicial de una partícula es $x_\circ = -5$ m para $t_\circ = 0$ y con velocidad vx = 3 m/s. Después de 3 s, se adiciona una velocidad v1 = 3 m/s transcurriendo 5 s. Despupes nuevamente adquiere vx = 3 m/s por 2 s. Finalmente se adiciona una velocidad de v2 = −5 m/s y la particula avanza por 5 s. Encuentre la posici´on de la partícula.

[31]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema

acelerations =[
    [Vector(value=Coordenate(x=0,y=3)),3],
    [Vector(value=Coordenate(x=0,y=3)),5],
    [Vector(value=Coordenate(x=0,y=3)),2],
    [Vector(value=Coordenate(x=0,y=-5)),5]
]

t0 = 0                                          # tiempo inicial
y0 = Vector(value=Coordenate(x=0,y=-5))              # punto inicial
v0 = Vector(value=Coordenate(x=1,y=0), origin=y0.destiny)  # vector velocidad
```

```
yf = y0
vy = []
vx = []
vs = []
dx = []
dy = []
dx.append(yf.destiny.x)
dy.append(yf.destiny.y)

for a in acelerations:
    v0 = v0 + a[0]
    a[0].setOrigin(Coordenate(x = yf.x, y = 0))
    yfy = yf.y
    yf =  yf + v0.mulEscalar(a[1])
    yf.setOrigin(v0.origin)
    dx.append(yf.destiny.x)
    dy.append(yf.destiny.y)
    vx = vx + list(range(yfy, yf.y))
    vy = vy + [v0.y]*v0.mulEscalar(a[1]).y
    vs = vs + [v0.y]*a[1]

axis[1][0].grid()
vector2D(axis[1][0], yf)
axis[1][0].plot(dx, dy)
point2D(axis[1][0], yf.destiny)
point2D(axis[1][0], y0.destiny)
axis[1][0].set(ylabel = 'Desplazamiento', xlabel='Tiempo')

axis[0][0].plot(np.diff(vs))
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')

axis[0][1].plot(vx,vy)
axis[0][1].grid()
axis[0][1].set(ylabel = 'Velocidad', xlabel='Desplazamiento')

axis[1][1].plot(vs)
axis[1][1].set(ylabel = 'Velocidad', xlabel='Tiempo')
axis[1][1].grid()

print("Resultado:\t{}".format(yf.y))
```

```
Resultado:      72
```



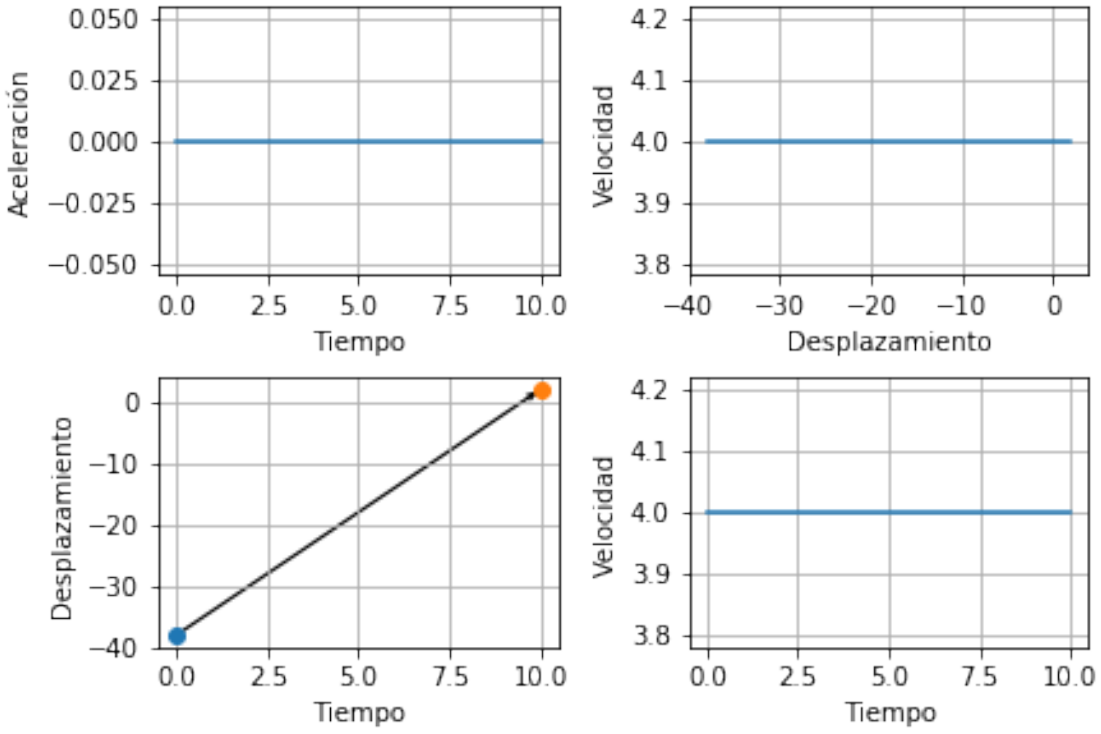
4.1.7 Ejercicio 7

Una partícula, llega a una posición final x$_\circ$ = 2 m en 10 s con vx = 4 m/s. Encuentre la posición inicial cuando t$_\circ$ = 0.

[32]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema

t0 = 0                                              # tiempo inicial
yf = Vector(value=Coordenate(x=10,y=2))                  # punto inicial
v0 = Vector(value=Coordenate(x=1,y=4))  # vector velocidad
tf = 10                                             # vector del punto inicial
pv = v0.mulEscalar(tf - t0)                        # vector velocidad de la␣
 ↪particula por el tiempo total
y0 = yf - pv                                 # vector resultante del desplazamiento
a0 = Vector()
pv.setOrigin(y0.destiny)

#point2D(axis[1][0], Coordenate())
#vector2D(axis[1][0], y0)
#vector2D(axis[1][0], yf)


point2D(axis[1][0], y0.destiny)
```

```
point2D(axis[1][0], yf.destiny)
vector2D(axis[1][0], pv)
axis[1][0].set(ylabel = 'Desplazamiento', xlabel='Tiempo')
axis[1][0].grid()


axis[1][1].plot([t0,tf],[v0.y , v0.y])
axis[1][1].set(ylabel = 'Velocidad', xlabel='Tiempo')
axis[1][1].grid()

axis[0][0].plot([t0,tf], [a0.x, a0.x])
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')

axis[0][1].plot([y0.y, yf.y], [v0.y, v0.y])
axis[0][1].grid()
axis[0][1].set(ylabel = 'Velocidad', xlabel='Desplazamiento')


print("Resultado:\t{}".format(y0.y))
```

```
Resultado:      -38
```

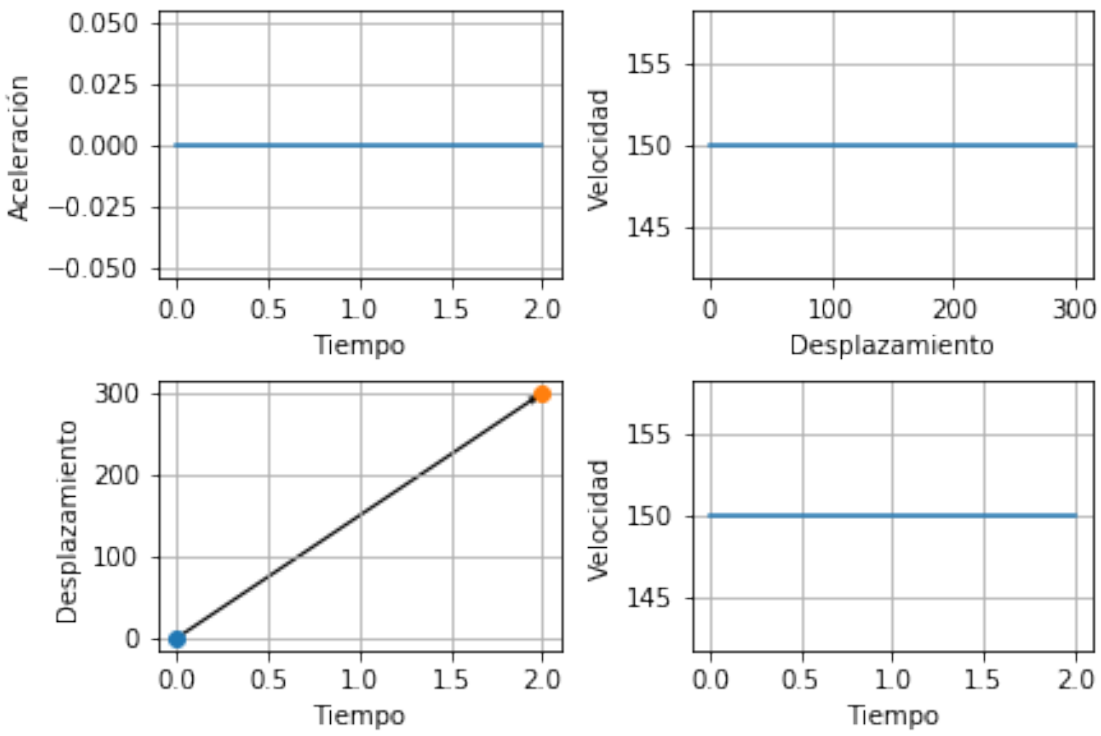


4.1.8 Ejercicio 8

Un automóvil parte del reposo desde $x_\circ = 0$ y pasa por $x = 300$ m en 2 s. Determine la velocidad.

[33]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema

t0 = 0                                          # tiempo inicial
y0 = Vector(value=Coordenate(x=0,y=0))               # punto inicial
tf = 2                                         # vector del punto inicial
yf = Vector(value=Coordenate(x=2,y=300))                                # vector␣
↪resultante del desplazamiento
v0 = (yf-y0)  # vector velocidad
v0 = v0.mulEscalar(1/v0.x)
a0 = Vector()
pv.setOrigin(y0.destiny)


point2D(axis[1][0], y0.destiny)
point2D(axis[1][0], yf.destiny)
vector2D(axis[1][0], yf)
axis[1][0].set(ylabel = 'Desplazamiento', xlabel='Tiempo')
axis[1][0].grid()


axis[1][1].plot([t0,tf],[v0.y , v0.y])
axis[1][1].set(ylabel = 'Velocidad', xlabel='Tiempo')
axis[1][1].grid()

axis[0][0].plot([t0,tf], [a0.x, a0.x])
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')

axis[0][1].plot([y0.y, yf.y], [v0.y, v0.y])
axis[0][1].grid()
axis[0][1].set(ylabel = 'Velocidad', xlabel='Desplazamiento')


print("Resultado:\t{}".format(v0.y))
```

```
Resultado:      150.0
```



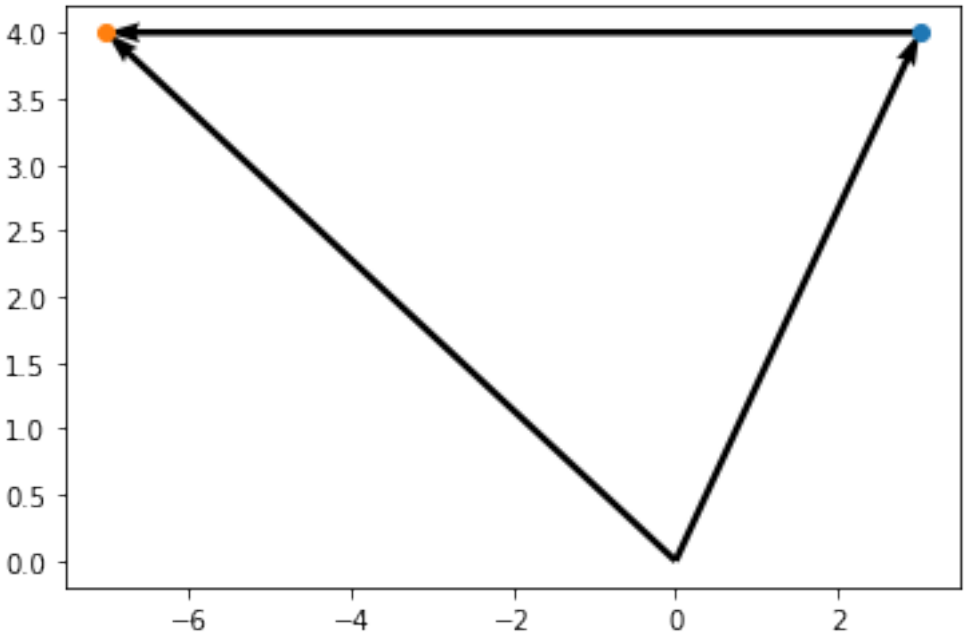
4.2 Velocidad constante en dos dimensiones

4.2.1 Ejercicio 1

Una partícula parte del reposo desde r = (3i+ 4j) m con una velocidad v = −2i m/s. Encuentre su posición final y desplazamiento cuando t = 5 s.

[14]:
```
fig, ax = plt.subplots()

r = Vector(value=Coordenate(x=3,y=4))
v = Vector(value=Coordenate(x=-2,y=0), origin=r.destiny)
vp = v.mulEscalar(5)
d = r+vp
vector2D(ax, r)
vector2D(ax, vp)
vector2D(ax, d)
point2D(ax, r.destiny)
point2D(ax, vp.destiny)

print("Resultado\t:\t[{};{}]".format(d.x , d.y))
```

```
Resultado       :       [-7;4]
```

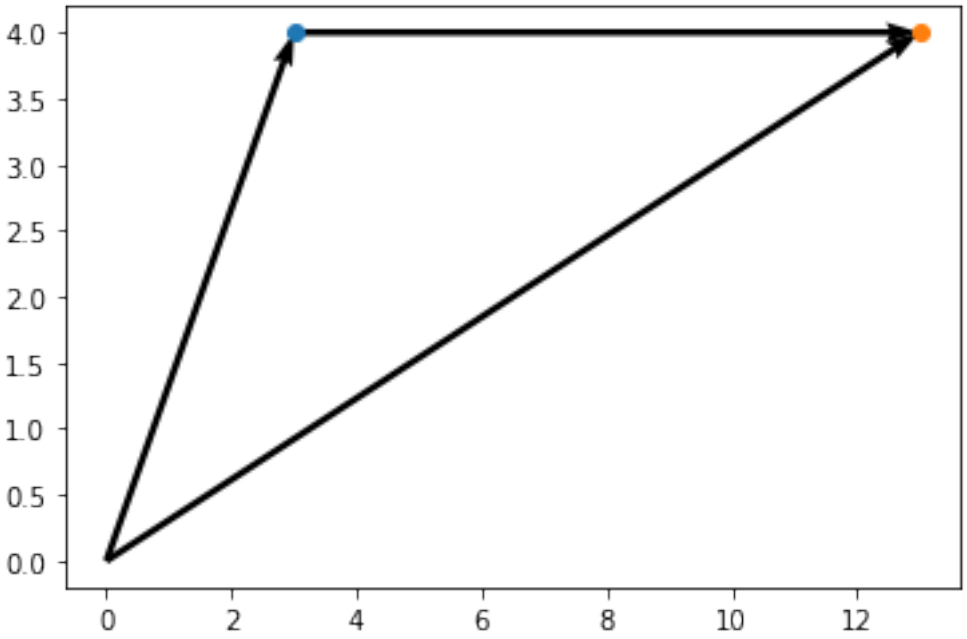


4.2.2 Ejercicio 2

Una partícula parte del reposo desde r = (3i+ 4j) m con una velocidad v = 2j m/s. Encuentre su posición final y desplazamiento cuando t = 5 s.

[15]:
```
fig, ax = plt.subplots()

r = Vector(value=Coordenate(x=3,y=4))
v = Vector(value=Coordenate(x=2,y=0), origin=r.destiny)
vp = v.mulEscalar(5)
d = r+vp
vector2D(ax, r)
vector2D(ax, vp)
vector2D(ax, d)
point2D(ax, r.destiny)
point2D(ax, vp.destiny)

print("Resultado\t:\t[{};{}]".format(d.x , d.y))
```

```
Resultado       :       [13;4]
```

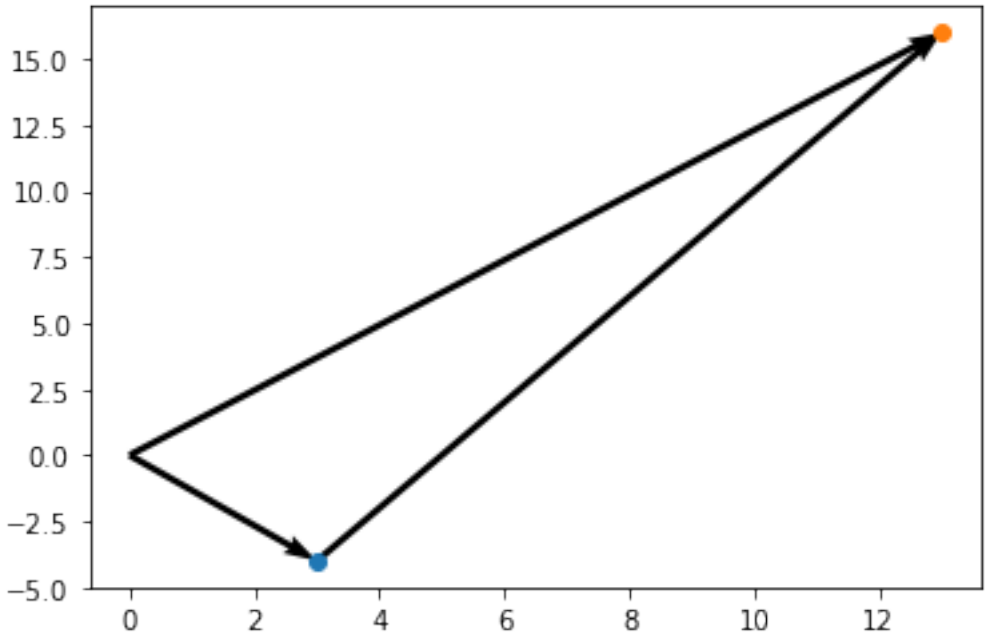


4.2.3 Ejercicio 3

Una partícula parte del reposo desde r = (3i − 4j) m con una velocidad v = (2i + 4j) m/s. Encuentre su posición final y desplazamiento cuando t = 5 s.

[16]:
```
fig, ax = plt.subplots()

r = Vector(value=Coordenate(x=3,y=-4))
v = Vector(value=Coordenate(x=2,y=4), origin=r.destiny)
vp = v.mulEscalar(5)
d = r+vp
vector2D(ax, r)
vector2D(ax, vp)
vector2D(ax, d)
point2D(ax, r.destiny)
point2D(ax, vp.destiny)

print("Resultado\t:\t[{};{}]".format(d.x , d.y))
```

```
Resultado       :       [13;16]
```

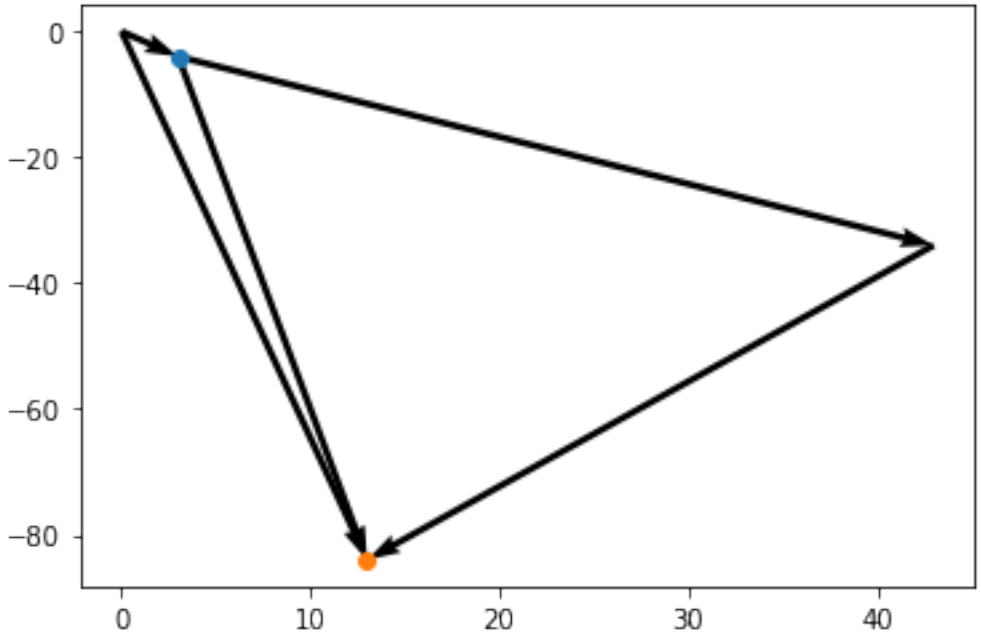


4.2.4 Ejercicio 4

Una partícula parte del reposo desde r = (3i−4j) m con una velocidad v = (4i−3j) m/s. Cuando inicia su movimiento se presenta una velocidad del viento vv = (−3i + 5j) m/s. Encuentre su posición final y desplazamiento cuando t = 10 s.

[17]:
```
fig, ax = plt.subplots()

r = Vector(value=Coordenate(x=3,y=-4))
v = Vector(value=Coordenate(x=4,y=-3), origin=r.destiny)
vp = v.mulEscalar(10)
vp.setOrigin(r.destiny)
vv = Vector(value=Coordenate(x=-3,y=-5), origin=vp.destiny)
vvp = vv.mulEscalar(10)
v_vv = vp + vvp
d = r+vp + vvp

vector2D(ax, r)
vector2D(ax, vp)
vector2D(ax, vvp)
vector2D(ax, v_vv)
vector2D(ax, d)
point2D(ax, r.destiny)
point2D(ax, d.destiny)

print("Resultado\t:\t[{};{}]".format(d.x , d.y))
```

```
Resultado       :       [13;-84]
```



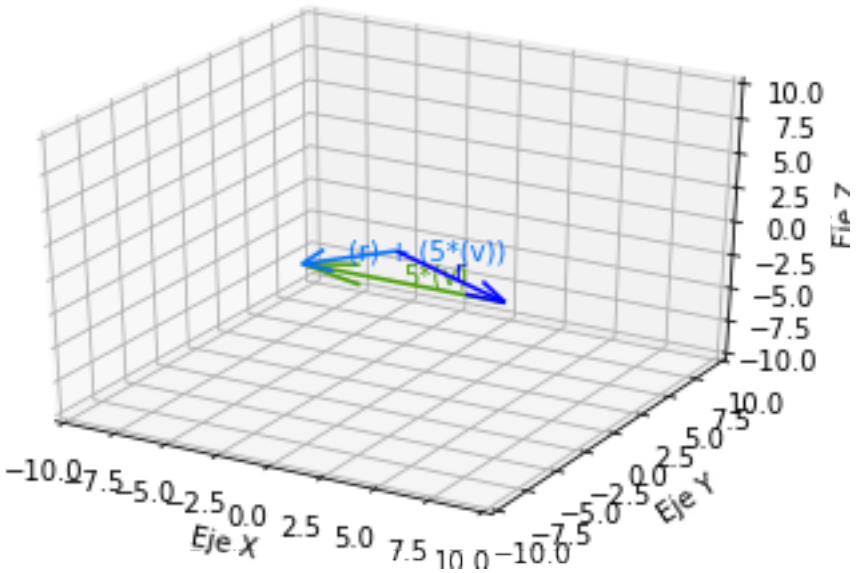
4.3 Velocidad constante en tres dimensiones

4.3.1 Ejericio 1

Una partícula parte del reposo desde r = (3i + 4j + 5k) m con una velocidad v = −2~i m/s. Encuentre su posición final y desplazamiento cuando t = 5 s.

[18]:
```
r = Vector(value=Coordenate(x = 3, y = 4, z = -5), label='r')
v = Vector(value=Coordenate(x = -2), origin=r.destiny, label='v')
vp = v.mulEscalar(5)
d = r + vp


g = Graphic3D()
g.vector(r)
g.vector(vp)
g.vector(d)

print("Resultando\t:\t{}".format(d))
```

```
Resultando      :       [-7.00; 4.00; -5.00]
```

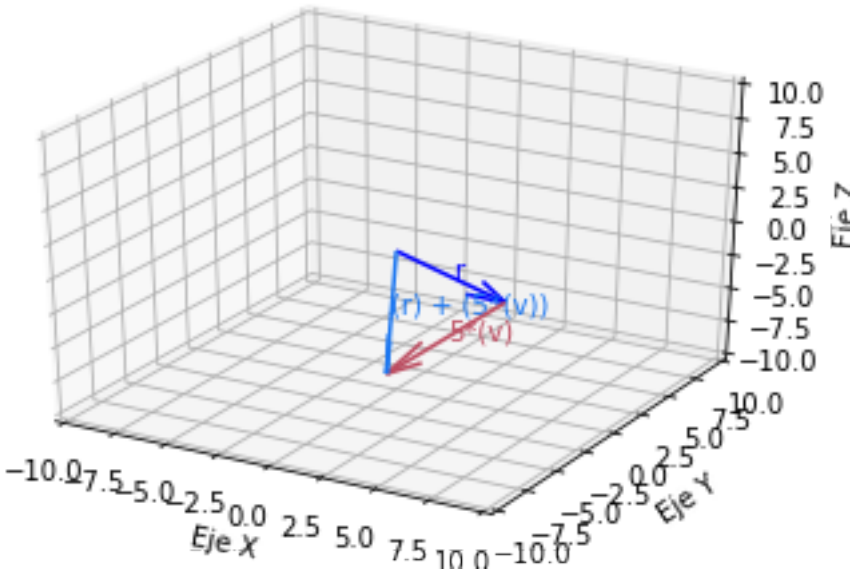


4.3.2 Ejericio 2

Una partícula parte del reposo desde r = (3i + 4j − 5k) m con una velocidad v = 2j m/s. Encuentre su posición final y desplazamiento cuando t = 5 s.

[19]:
```
r = Vector(value=Coordenate(x = 3, y = 4, z = -5), label='r')
v = Vector(value=Coordenate(y = -2), origin=r.destiny, label='v')
vp = v.mulEscalar(5)
d = r + vp


g = Graphic3D()
g.vector(r)
g.vector(vp)
g.vector(d)

print("Resultando\t:\t{}".format(d))
```

```
Resultando      :       [3.00; -6.00; -5.00]
```

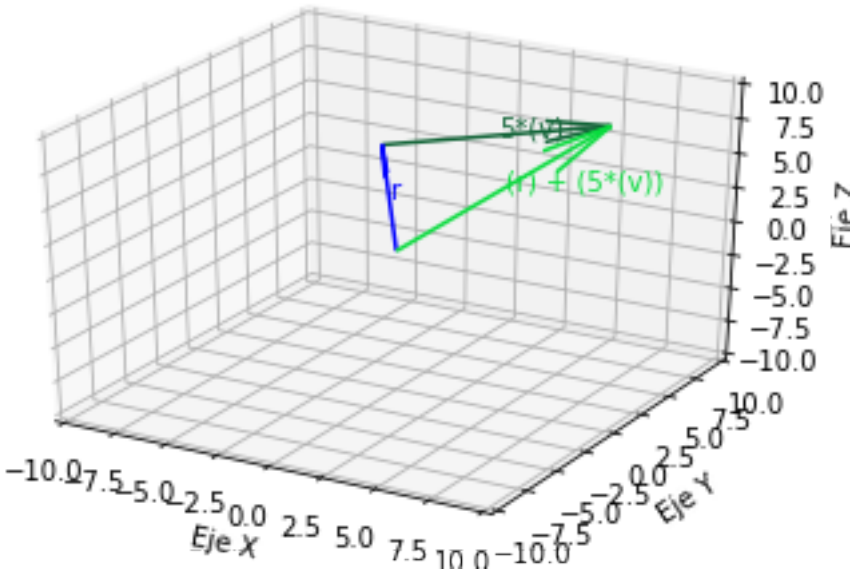


4.3.3 Ejericio 3

Una partícula parte del reposo desde r = (−3i + 4j + 5k) m con una velocidad v = 2k m/s. Encuentre su posición final y desplazamiento cuando t = 5 s.

[20]:
```
r = Vector(value=Coordenate(x = -3, y = 4, z = +5), label='r')
v = Vector(value=Coordenate(x = 1, y=1), label='v').resize(2)
v.setLabel('v')
v.setOrigin(r.destiny)
vp = v.mulEscalar(5)
d = r + vp


g = Graphic3D()
g.vector(r)
g.vector(vp)
g.vector(d)

print("Resultando\t:\t{}".format(d))
```

```
Resultando      :       [4.07; 11.07; 5.00]
```

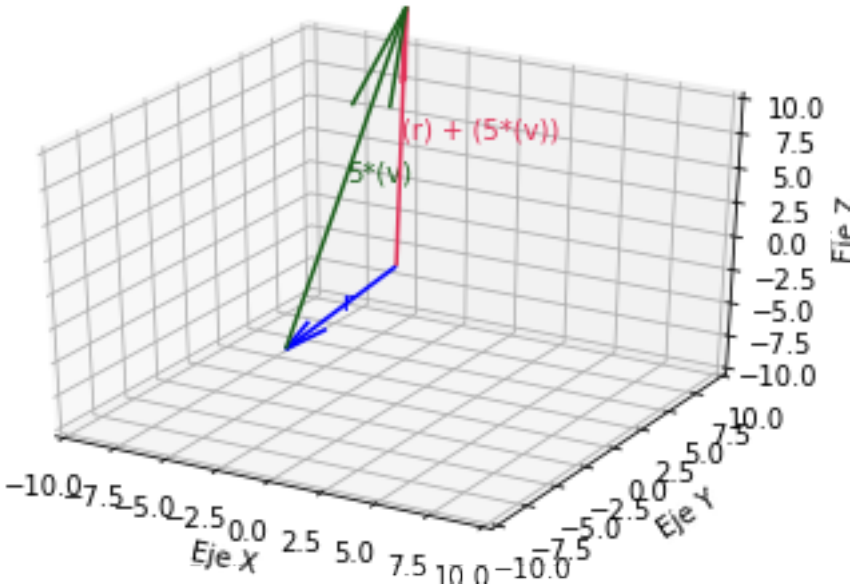


4.3.4 Ejericio 4

Una partícula parte del reposo desde r = (−3i − 4j − 5k) m con una velocidad v = (2j + 4k)m/s. Encuentre su posición final y desplazamiento cuando t = 5 s.

[21]:
```
r = Vector(value=Coordenate(x = -3, y = -4, z = -5), label='r')
v = Vector(value=Coordenate(y = 2, z=4), label='v', origin=r.destiny)
vp = v.mulEscalar(5)
d = r + vp


g = Graphic3D()
g.vector(r)
g.vector(vp)
g.vector(d)

print("Resultando\t:\t{}".format(d))
```

```
Resultando      :       [-3.00; 6.00; 15.00]
```

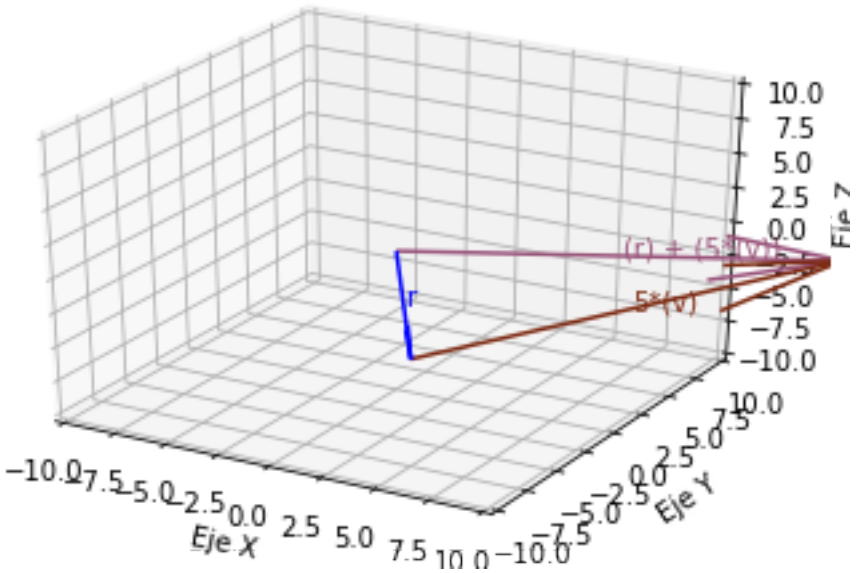


4.3.5 Ejericio 5

Una partícula parte del reposo desde r = (3i − 4j − 5k) m con una velocidad v = (2i + 4j) m/s. Encuentre su posición final y desplazamiento cuando t = 5 s

[22]:
```
r = Vector(value=Coordenate(x = 3, y = -4, z = -5), label='r')
v = Vector(value=Coordenate(x = 2, y=4), label='v', origin=r.destiny)
vp = v.mulEscalar(5)
d = r + vp


g = Graphic3D()
g.vector(r)
g.vector(vp)
g.vector(d)

print("Resultando\t:\t{}".format(d))
```

```
Resultando      :       [13.00; 16.00; -5.00]
```

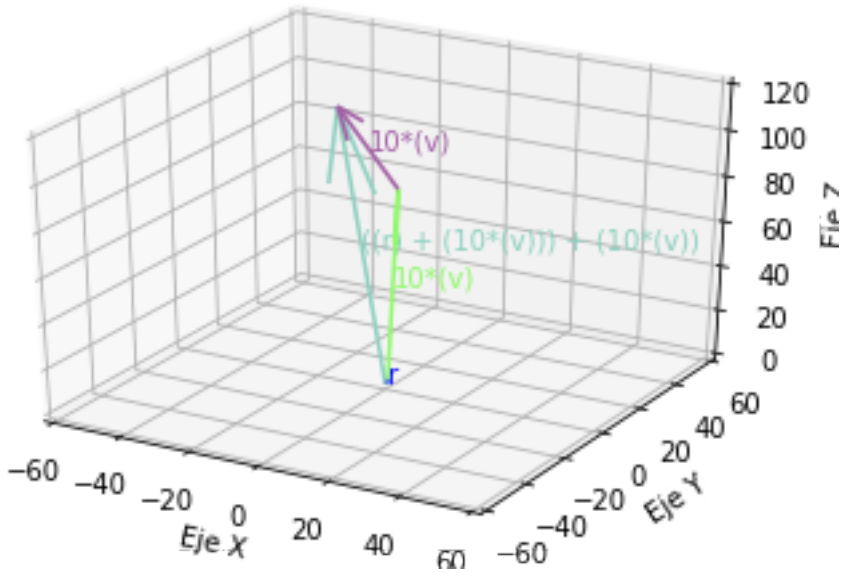


4.3.6 Ejericio 6

Una partícula parte del reposo desde r = (3i−4j + 5k) m con una velocidad v = (−2i+ 4j + 6k) m/s. Cuando inicia su movimiento se presenta una velocidad del viento vv = (−3j + 5k) m/s. Encuentre su posición final y desplazamiento cuando t = 10 s

[23]:
```
r = Vector(value=Coordenate(x = 3, y = -4, z = 5), label='r')
v = Vector(value=Coordenate(x = -2, y=4, z=6), label='v', origin=r.destiny)
vp = v.mulEscalar(10)
vv = Vector(value=Coordenate(y=-3, z=5), label='v', origin=vp.destiny)
vvp = vv.mulEscalar(10)
d = r + vp + vvp


g = Graphic3D(zmin = 0, zmax =120, ymin = -60, ymax =60, xmin = -60, xmax =60)
g.vector(r)
g.vector(vp)
g.vector(d)
g.vector(vvp)

print("Resultando\t:\t{}".format(d))
```

```
Resultando      :       [-17.00; 6.00; 115.00]
```



5 Aceleración constante

5.1 Aceleración constante en una dimensión. Caída libre

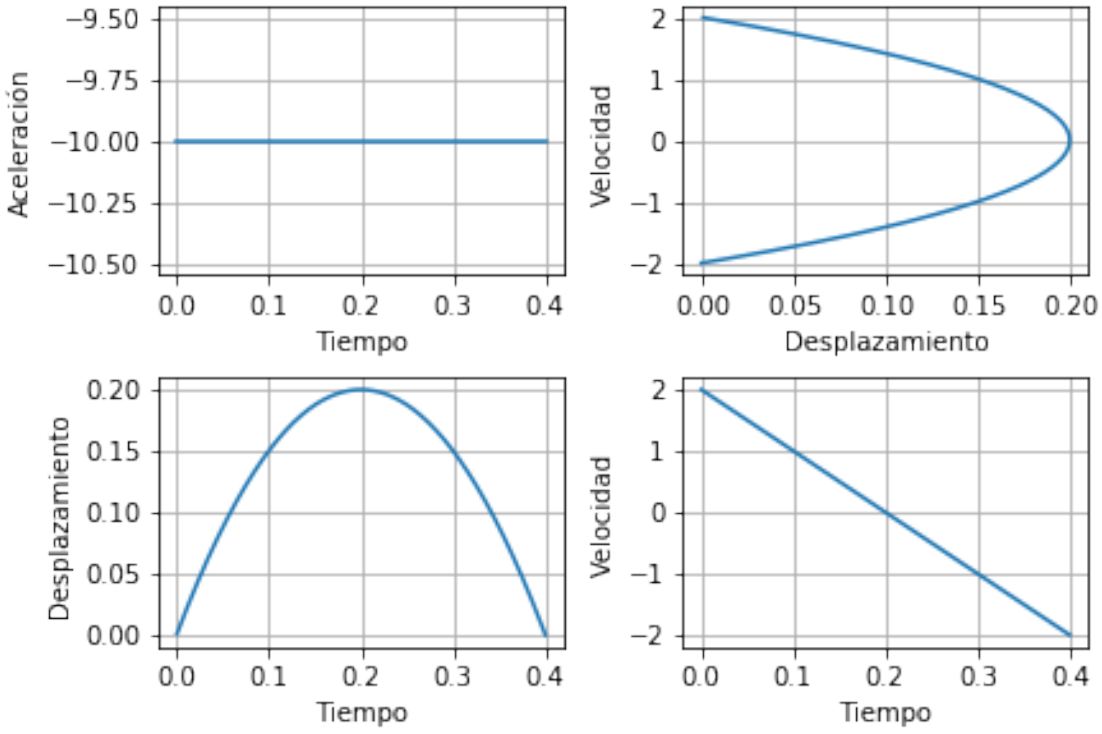
5.1.1 Ejericio 1

Una partícula se lanza desde el suelo hacia arriba con una velocidad inicial vyi = 2 m/s. Haga los diagramas a − t, v − t, x − t y v − x cuando la partícula llegue al suelo.

[34]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema
a = Vector(value=Coordenate(x=1, y=-10))

t0 = 0                                          # tiempo inicial
y0 = Vector(value=Coordenate(x=0,y=0))              # punto inicial
v0 = Vector(value=Coordenate(x=0,y=2))  # vector velocidad
tf = 0.4                                        # vector del punto inicial
pv.setOrigin(y0.destiny)
h = 0.0005
yf = y0
axis[0][0].plot([t0,tf], [a.y, a.y])
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')

t = [ 0 ]
vy = [ v0.y ]
dy = [ 0 ]
```

```
for i in np.arange(0,tf,h):
    v0 = v0 + a.mulEscalar(h)
    yf = yf + v0.mulEscalar(h)
    t.append(i+h)
    vy.append(v0.y)
    dy.append(yf.y)

axis[1][1].plot(t,vy)
axis[1][1].set(ylabel = 'Velocidad', xlabel='Tiempo')
axis[1][1].grid()

axis[1][0].plot(t, dy)
axis[1][0].set(ylabel = 'Desplazamiento', xlabel='Tiempo')
axis[1][0].grid()

axis[0][1].plot(dy, vy)
axis[0][1].grid()
axis[0][1].set(ylabel = 'Velocidad', xlabel='Desplazamiento')

print("Resultado:\t{}".format(yf.y))
```

```
Resultado:      -0.0009999999999918301
```



6 Anexos

6.1 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()
```

[ ]:
```

```

6.2 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__)
```

6.3 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
```

[ ]: