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
import sys
import random
```

2 Importando Modelos

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

3 Constantes

[4]:
```
MAX_INT = 100
MIN_INT = -MAX_INT
```

4 Metodos

4.1 Obtener Signo

[5]:
```
def getSigno(x):
    if x == 0:
        return 0
    return x/abs(x)
```

4.2 Graficar Vector 2D

[6]:
```
def vector2D(axis, vector):
    axis.quiver(vector.origin.x, vector.origin.y,  vector.x,vector.y ,␣
 ↪angles='xy', scale_units='xy', scale=1)
```

4.3 Graficar Punto 2D

[7]:
```
def point2D(axis, coordenate):
    axis.plot(coordenate.x, coordenate.y, marker="o", )
```

5 a = cte

5.1 Movimiento en una dimensión. Caída libre g = −10 m/s2

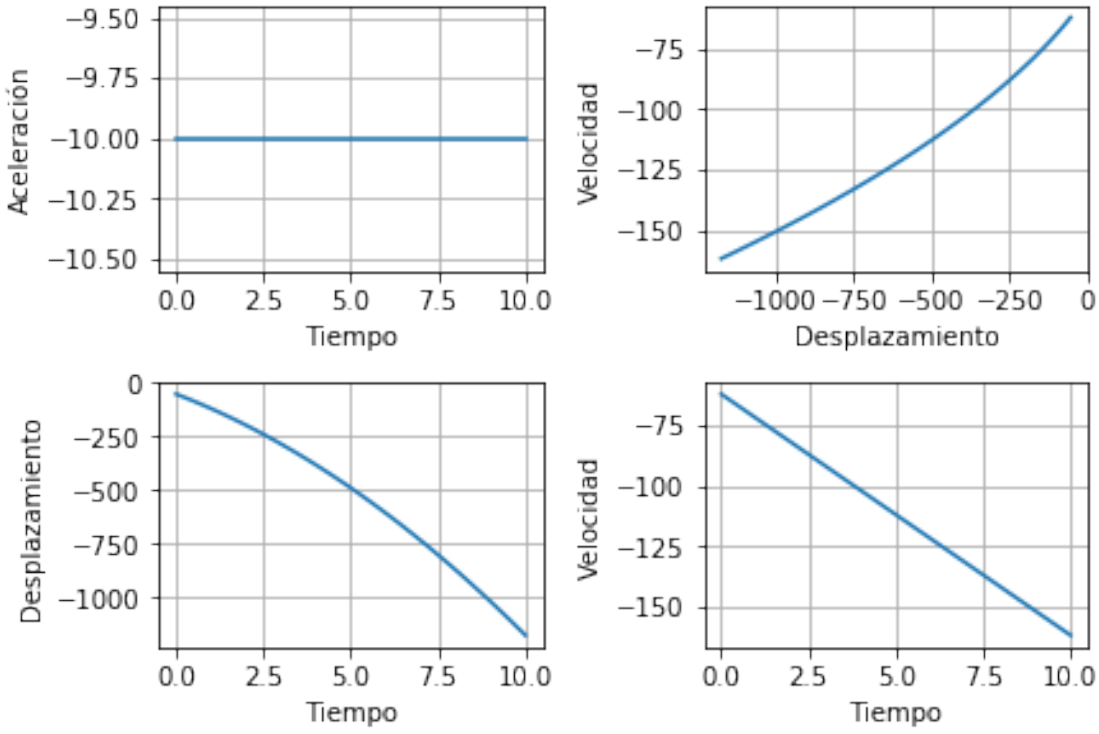
5.1.1 Sea una partícula, que tiene una posición inicial y∘ y velocidad inicial vy∘ en un tiempo inicial t = 0. Determine la posición final de la partíicula para t = 10 s con g = −10 m/s2. Haga los diagramas a − t, v − t, x − t y v − x y por cada caso de un ejemplo real. Considere h = 0.1

y∘ < 0, vy∘ < 0

[8]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema
a = Vector(value=Coordenate(x=1, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=random.randint(MIN_INT, 0)))
v0 = Vector(value=Coordenate(x=0,y=random.randint(MIN_INT, 0)))


tf = 10
h = 0.1
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]

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)
    as_.append(a.y)


axis[0][0].plot(t, as_)
axis[0][0].grid()
```

```
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')


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("Posicion FInal\t\t:\t{}".format(yf.y))
```

```
Tiempo inicial          :       0
Punto Inicial           :       -52
Velocidad Inicial       :       -62
Tiempo Final            :       10
Posicion FInal          :       -1177.0000000000002
```

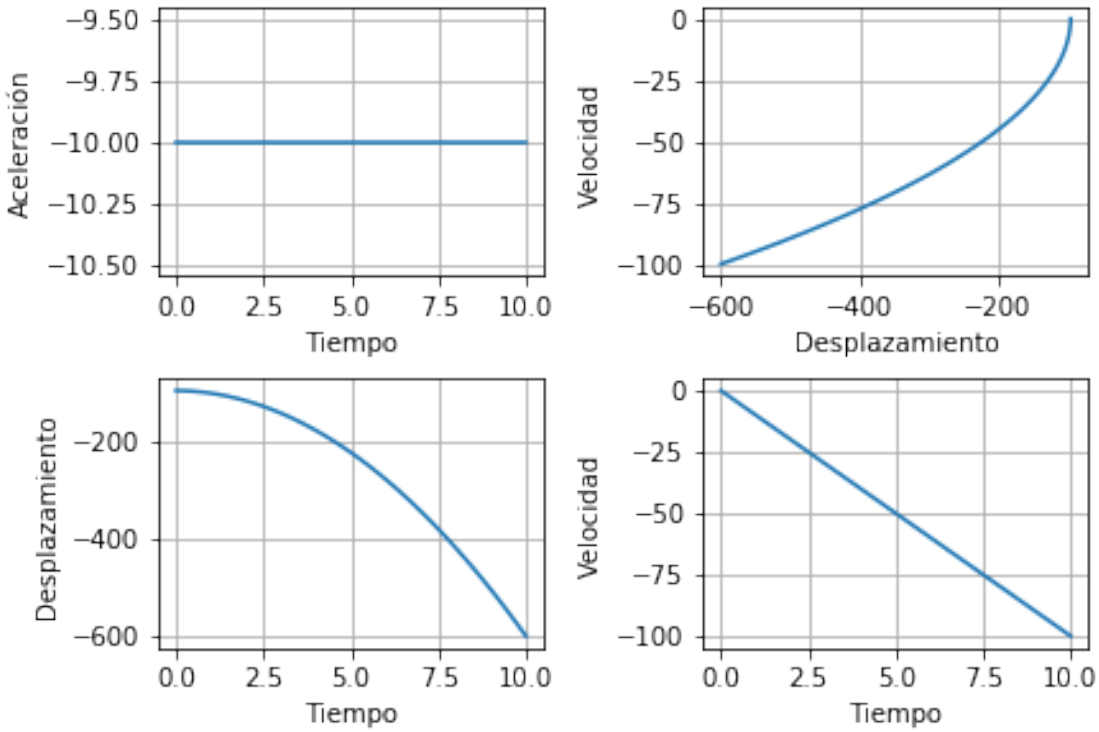


$\mathbf{y_\circ < 0, vy_\circ = 0}$

[9]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema
a = Vector(value=Coordenate(x=1, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=random.randint(MIN_INT, 0)))
v0 = Vector(value=Coordenate(x=0,y=0))



tf = 10
h = 0.1
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))



t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]

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)
    as_.append(a.y)



axis[0][0].plot(t, as_)
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')



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("Posicion FInal\t\t:\t{}".format(yf.y))
```

```
Tiempo inicial          :       0
Punto Inicial           :       -95
Velocidad Inicial       :       0
Tiempo Final            :       10
Posicion FInal          :       -600.0
```

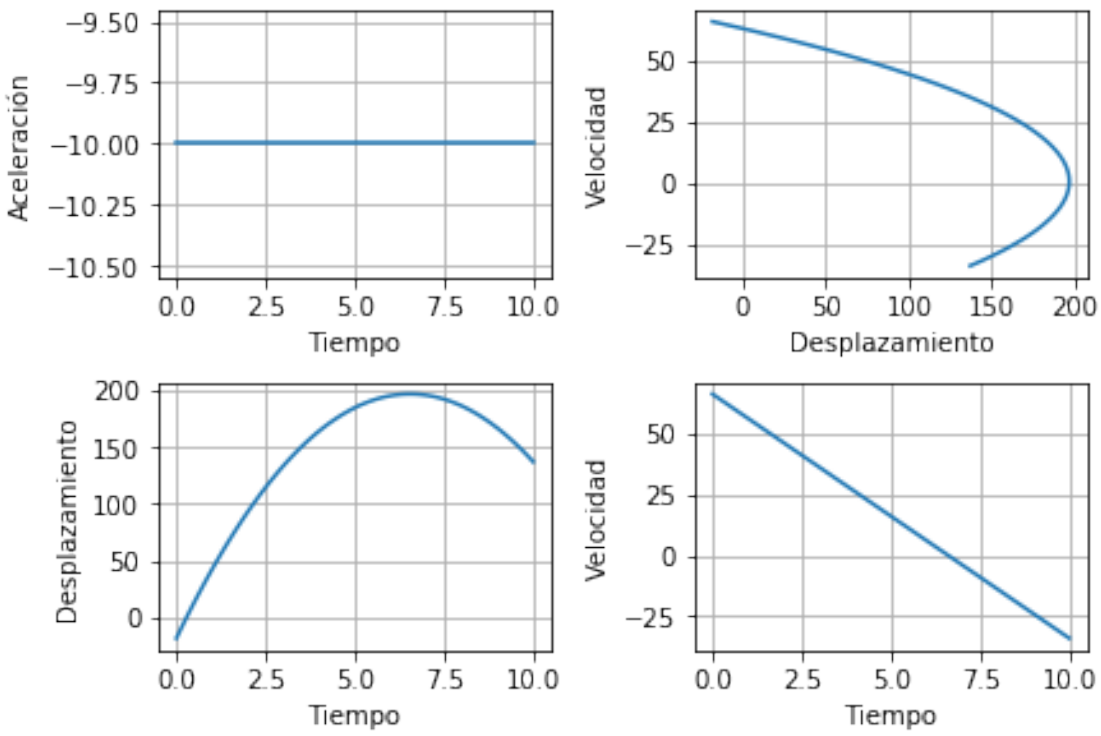


$y_0 < 0$, $vy_0 > 0$

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

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=random.randint(MIN_INT, 0)))
v0 = Vector(value=Coordenate(x=0,y=random.randint(0, MAX_INT)))


tf = 10
h = 0.1
yf = y0
```

```
print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]

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)
    as_.append(a.y)


axis[0][0].plot(t, as_)
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')


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("Posicion FInal\t\t:\t{}".format(yf.y))
```

```
Tiempo inicial          :       0
Punto Inicial           :       -18
Velocidad Inicial       :       66
Tiempo Final            :       10
Posicion FInal          :       137.0
```

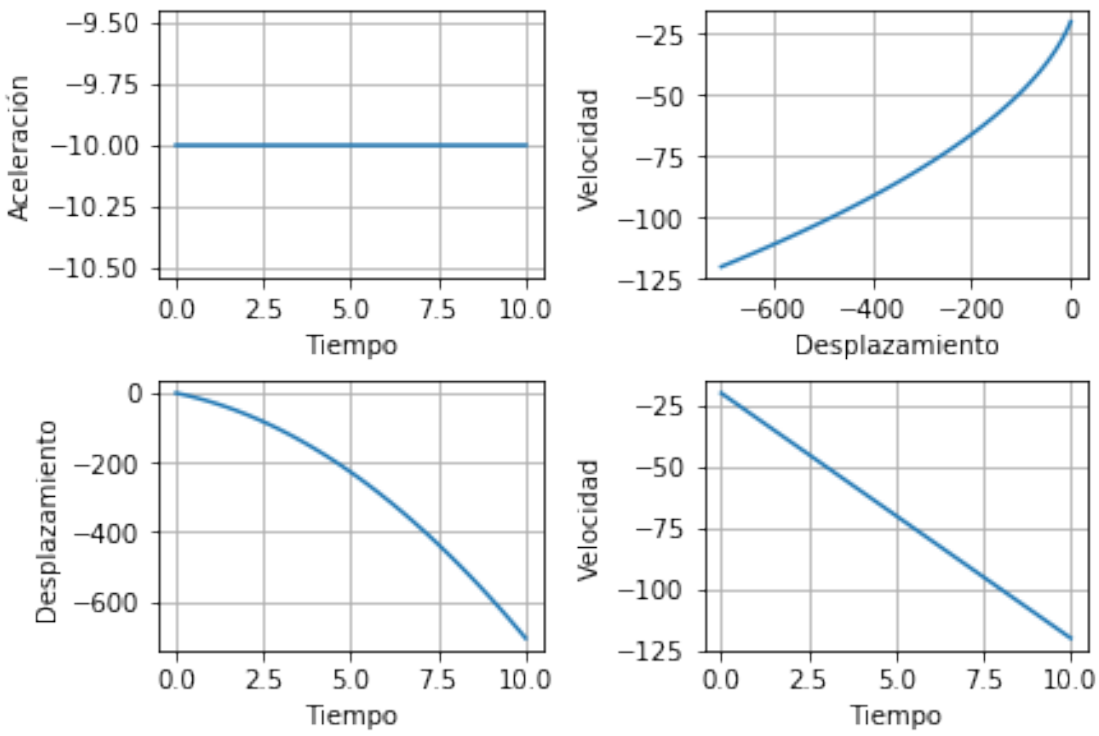


$\mathbf{y_\circ = 0, vy_\circ < 0}$

[11]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema
a = Vector(value=Coordenate(x=1, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=0))
v0 = Vector(value=Coordenate(x=0,y=random.randint(MIN_INT, 0)))


tf = 10
h = 0.1
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]
```

```
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)
    as_.append(a.y)


axis[0][0].plot(t, as_)
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')


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("Posicion FInal\t\t:\t{}".format(yf.y))
```

```
Tiempo inicial          :       0
Punto Inicial           :       0
Velocidad Inicial       :       -20
Tiempo Final            :       10
Posicion FInal          :       -705.0
```

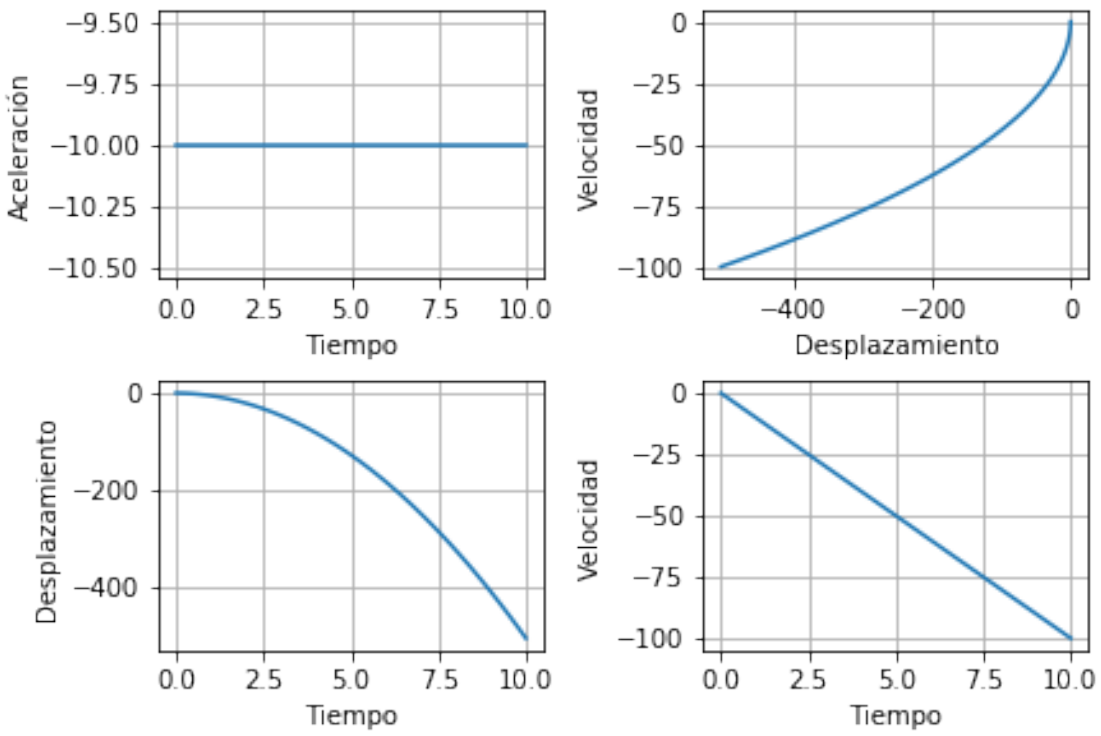


5.1.2 $y_\circ = 0$, $vy_\circ = 0$

[12]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema
a = Vector(value=Coordenate(x=1, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=0))
v0 = Vector(value=Coordenate(x=0,y=0))


tf = 10
h = 0.1
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]
```

```
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)
    as_.append(a.y)


axis[0][0].plot(t, as_)
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')


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("Posicion FInal\t\t:\t{}".format(yf.y))
```

```
Tiempo inicial          :       0
Punto Inicial           :       0
Velocidad Inicial       :       0
Tiempo Final            :       10
Posicion FInal          :       -505.0
```

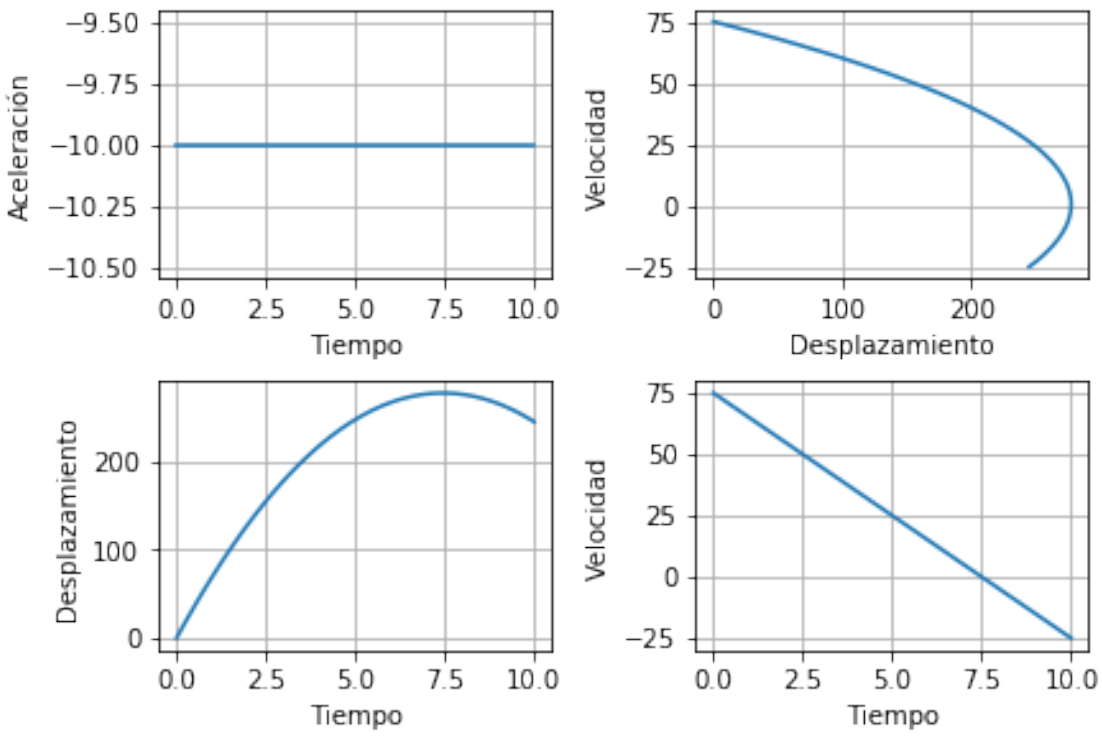


$\mathbf{y_\circ = 0, vy_\circ > 0}$

[13]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema
a = Vector(value=Coordenate(x=1, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=0))
v0 = Vector(value=Coordenate(x=0,y=random.randint(0, MAX_INT)))


tf = 10
h = 0.1
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]
```

```
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)
    as_.append(a.y)


axis[0][0].plot(t, as_)
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')


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("Posicion FInal\t\t:\t{}".format(yf.y))
```

```
Tiempo inicial          :       0
Punto Inicial           :       0
Velocidad Inicial       :       75
Tiempo Final            :       10
Posicion FInal          :       245.0
```

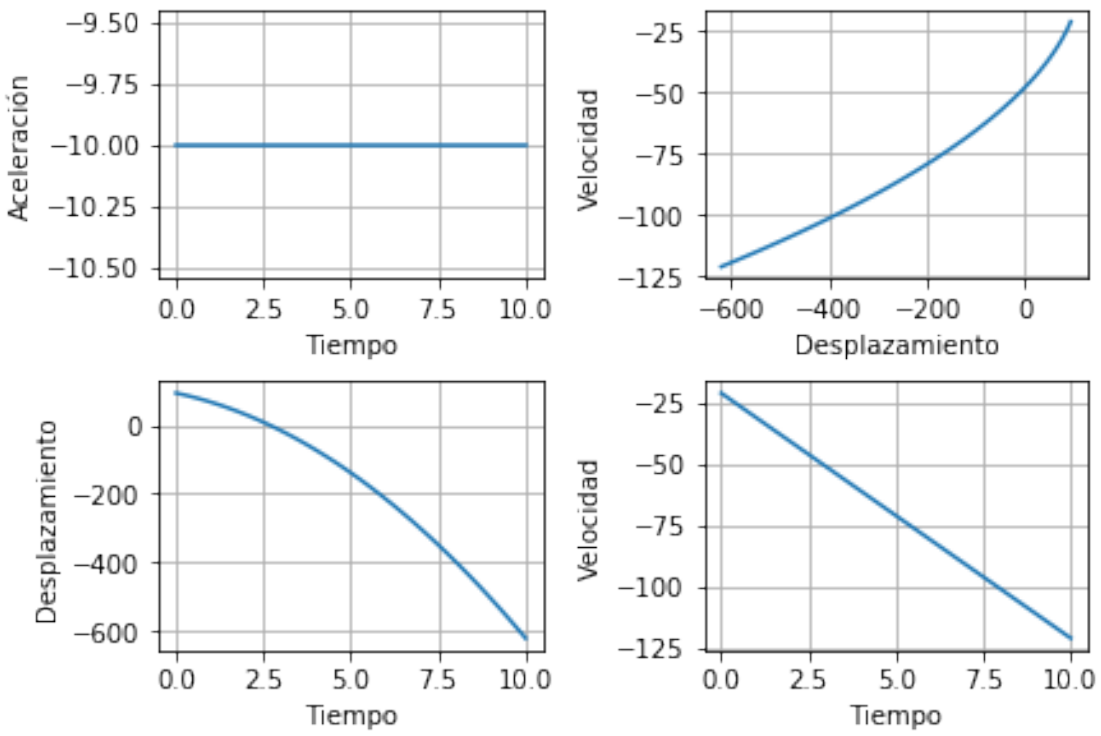


$y_\circ > 0$, $vy_\circ < 0$

[14]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema
a = Vector(value=Coordenate(x=1, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=random.randint(0, MAX_INT)))
v0 = Vector(value=Coordenate(x=0,y=random.randint(MIN_INT, 0)))


tf = 10
h = 0.1
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]
```

```
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)
    as_.append(a.y)


axis[0][0].plot(t, as_)
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')


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("Posicion FInal\t\t:\t{}".format(yf.y))
```

```
Tiempo inicial          :       0
Punto Inicial           :       94
Velocidad Inicial       :       -21
Tiempo Final            :       10
Posicion FInal          :       -621.0
```

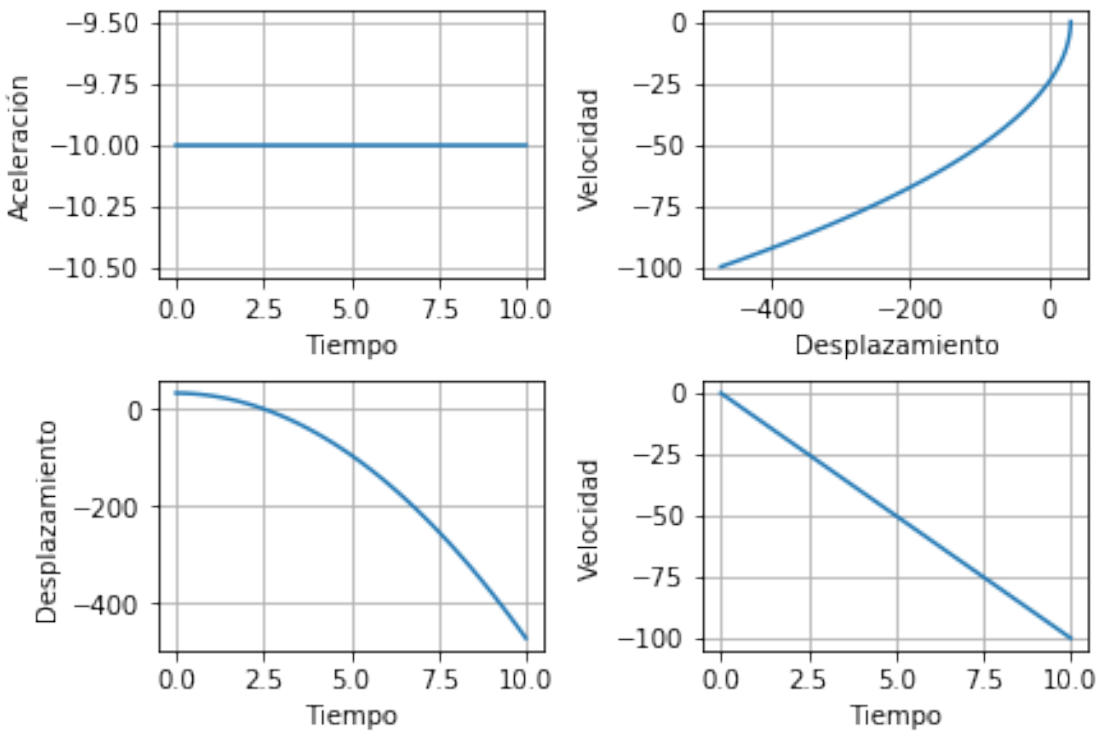


$\mathbf{y_\circ > 0, vy_\circ = 0}$

[15]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema
a = Vector(value=Coordenate(x=1, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=random.randint(0, MAX_INT)))
v0 = Vector(value=Coordenate(x=0,y=0))


tf = 10
h = 0.1
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]
```

```
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)
    as_.append(a.y)


axis[0][0].plot(t, as_)
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')


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("Posicion FInal\t\t:\t{}".format(yf.y))
```

```
Tiempo inicial          :       0
Punto Inicial           :       32
Velocidad Inicial       :       0
Tiempo Final            :       10
Posicion FInal          :       -473.0
```

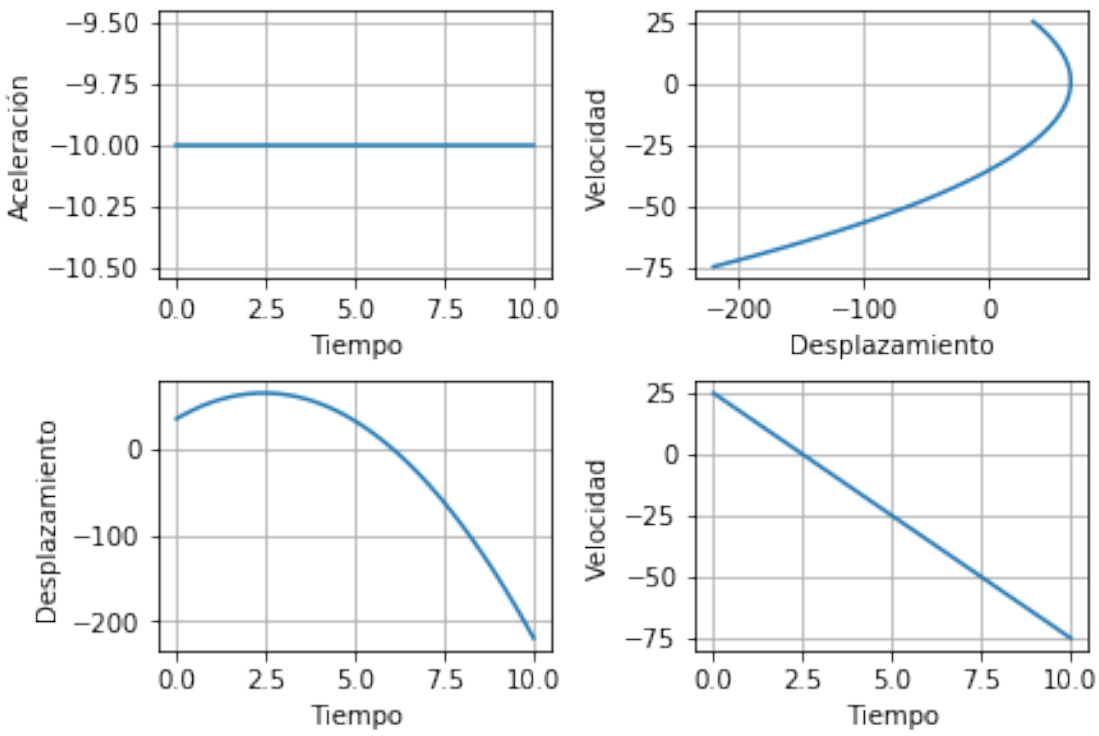


$y_\circ > 0$, $vy_\circ > 0$

[16]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema
a = Vector(value=Coordenate(x=1, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=random.randint(0, MAX_INT)))
v0 = Vector(value=Coordenate(x=0,y=random.randint(0, MAX_INT)))


tf = 10
h = 0.1
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]
```

```
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)
    as_.append(a.y)


axis[0][0].plot(t, as_)
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')


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("Posicion FInal\t\t:\t{}".format(yf.y))
```

```
Tiempo inicial          :       0
Punto Inicial           :       35
Velocidad Inicial       :       25
Tiempo Final            :       10
Posicion FInal          :       -220.00000000000003
```



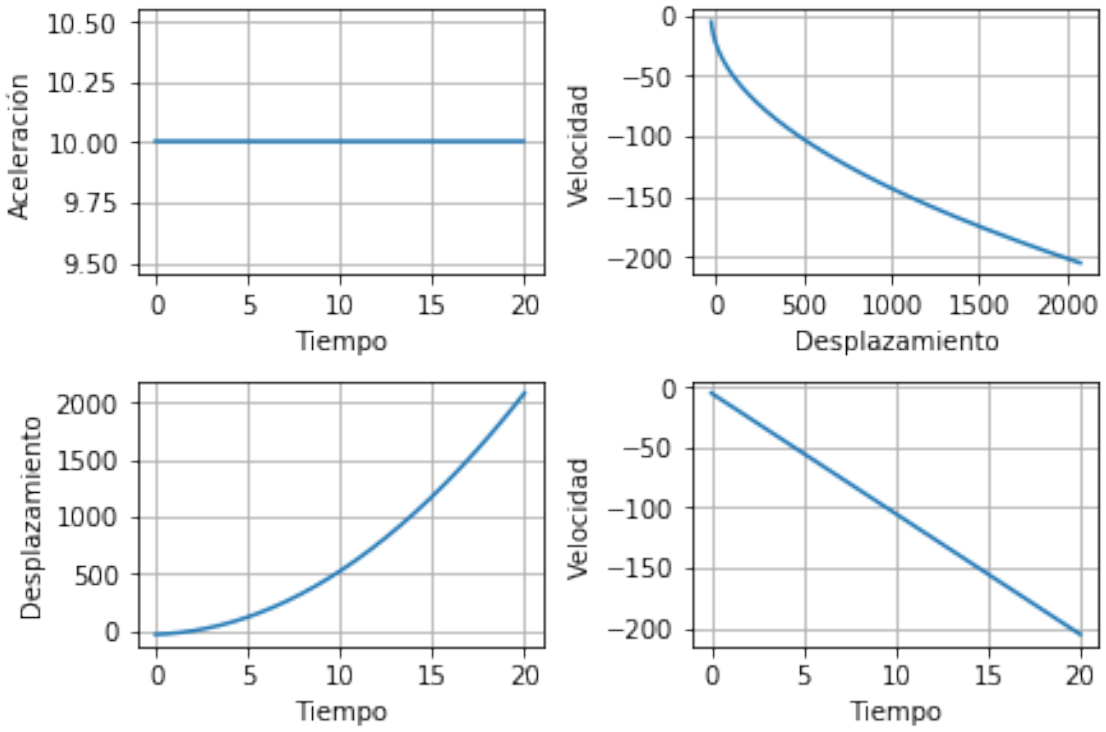
5.1.3 Una partícula se mueve, en el eje x con aceleración ax = 10 m/s2, hasta una posición final xf y velocidad final vxf que lo hace en un tiempo tf = 20 s. Encuentre la posición inicial x∘ y la vy∘. Haga los diagramas a − t, v − t, x − t y v − x y por cada caso de un ejemplo real. Considere h = 0.05

xf < 0, vxf < 0

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

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=random.randint(MIN_INT, 0)))
v0 = Vector(value=Coordenate(x=0,y=random.randint(MIN_INT, 0)))


tf = 20
h = 0.05
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
```

```
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]

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)
    as_.append(a.y)


axis[0][0].plot(t, as_)
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')


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("Posicion FInal\t\t:\t{}".format(yf.y))
```

```
Tiempo inicial          :       0
Punto Inicial           :       -26
Velocidad Inicial       :       -5
Tiempo Final            :       20
Posicion FInal          :       2079.0
```

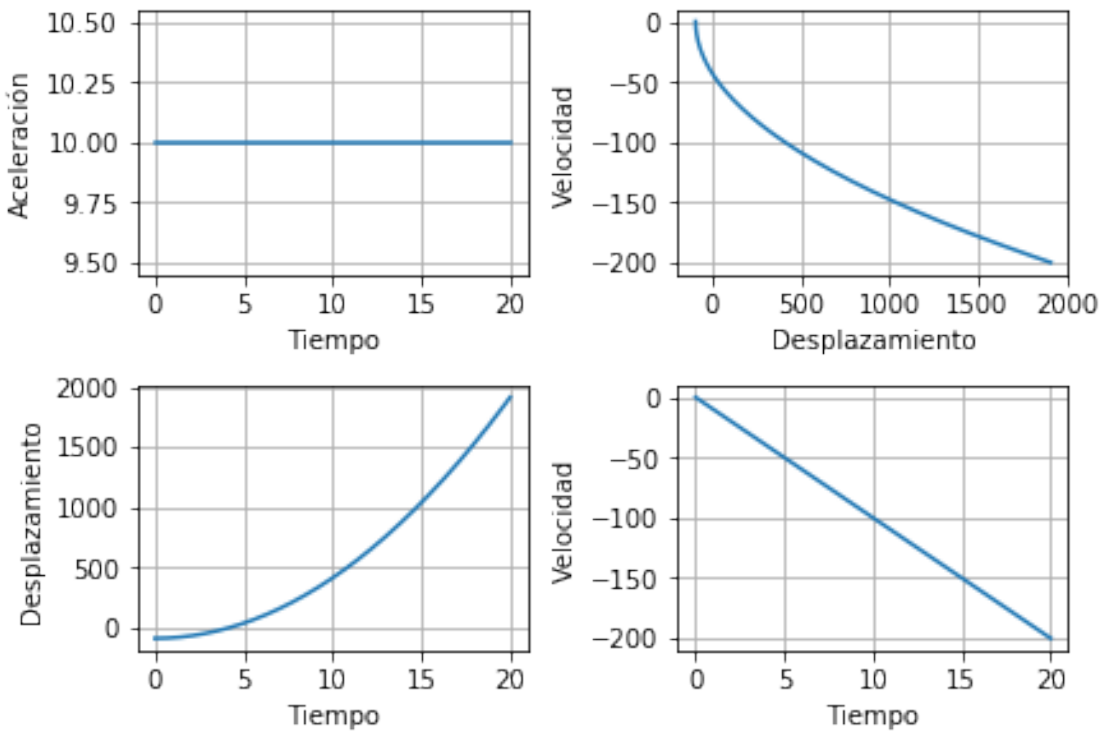


xf < 0, vxf = 0

[35]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema
a = Vector(value=Coordenate(x=1, y=10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=random.randint(MIN_INT, 0)))
v0 = Vector(value=Coordenate(x=0,y=0))


tf = 20
h = 0.05
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]
```

```
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)
    as_.append(a.y)


axis[0][0].plot(t, as_)
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')


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("Posicion FInal\t\t:\t{}".format(yf.y))
```

```
Tiempo inicial          :       0
Punto Inicial           :       -94
Velocidad Inicial       :       0
Tiempo Final            :       20
Posicion FInal          :       1911.0
```

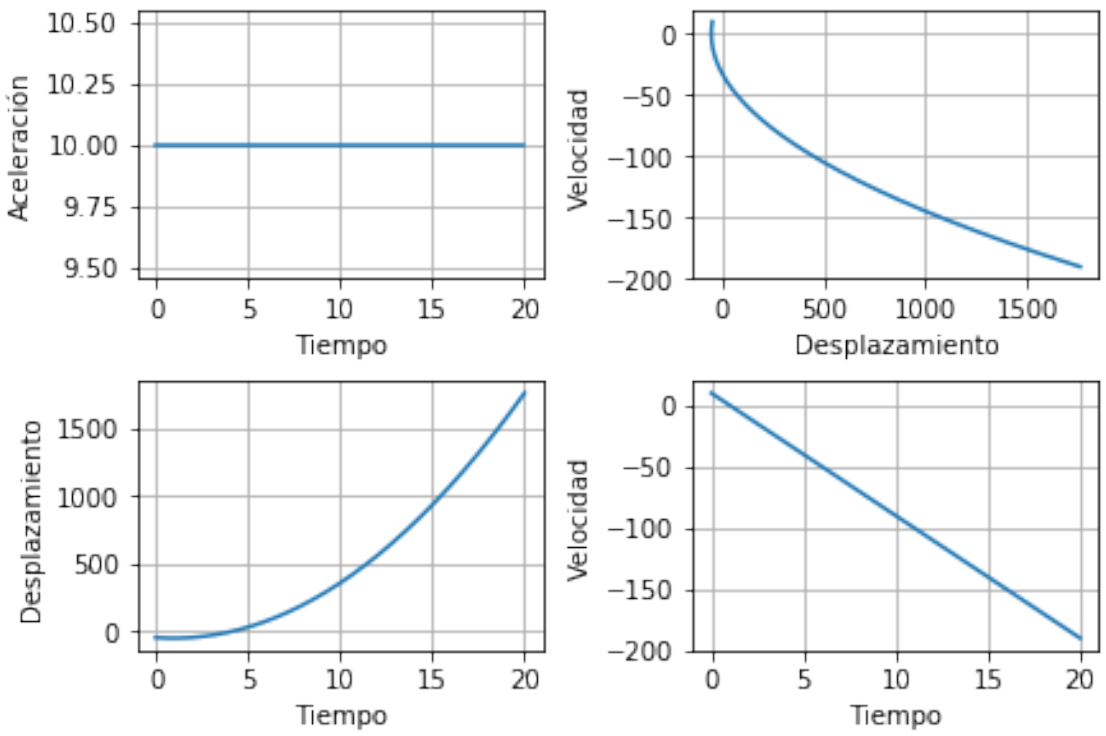


xf < 0, vxf > 0

[36]:
```
fig, axis = plt.subplots(2 , 2, constrained_layout=True) # Esquema
a = Vector(value=Coordenate(x=1, y=10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=random.randint(MIN_INT, 0)))
v0 = Vector(value=Coordenate(x=0,y=random.randint(0, MAX_INT)))


tf = 20
h = 0.05
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]
```

```
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)
    as_.append(a.y)


axis[0][0].plot(t, as_)
axis[0][0].grid()
axis[0][0].set(ylabel = 'Aceleración', xlabel='Tiempo')


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("Posicion FInal\t\t:\t{}".format(yf.y))
```

```
Tiempo inicial          :       0
Punto Inicial           :       -46
Velocidad Inicial       :       10
Tiempo Final            :       20
Posicion FInal          :       1758.9999999999998
```



5.1.4 Una partícula se mueve, en el eje x con aceleración ax = 10 m/s2, desde una posición inicial xi y velocidad inicial vxi que lo hace en un tiempo ti = 14 s. Encuentre la posición final xf y la vf. Haga el diagrama v − x − t. Considere h = 0.01

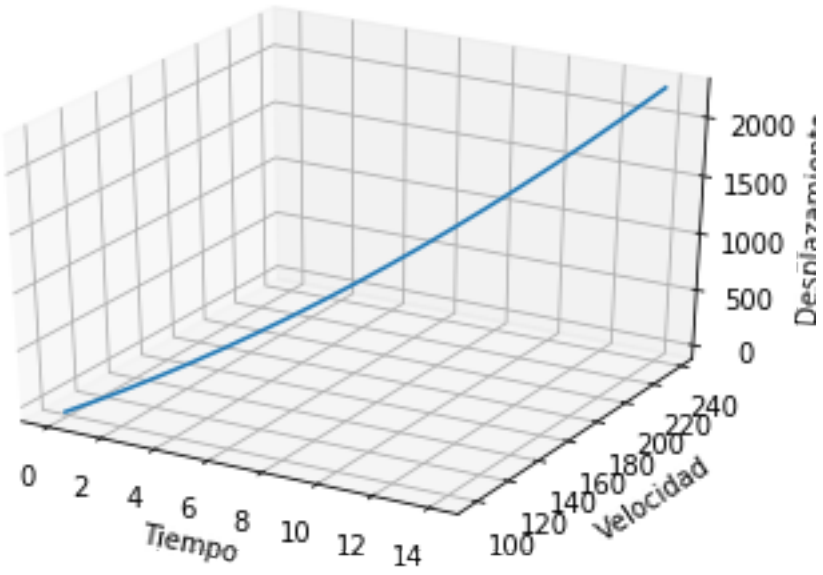
xi > 0, vxi < 0

[20]:
```
a = Vector(value=Coordenate(x=1, y=10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=random.randint(MIN_INT, 0)))
v0 = Vector(value=Coordenate(x=0,y=random.randint(0, MAX_INT)))


tf = 14
h = 0.01
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


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

```
dy = [ y0.y ]
as_ = [ a.y ]

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)
    as_.append(a.y)


print("Posicion FInal\t\t:\t{}\nVelocidad Final\t:\t{}".format(yf.y, v0.y))
fig = plt.figure()
ax3d = plt.axes(projection='3d')
ax3d.plot(t, vy, dy)
ax3d.set(xlabel='Tiempo', ylabel='Velocidad', zlabel='Desplazamiento')
```

```
Tiempo inicial          :       0
Punto Inicial           :       -75
Velocidad Inicial       :       98
Tiempo Final            :       14
Posicion FInal          :       2277.699999999945
Velocidad Final :       237.99999999999204
```

[20]:
```
[Text(0.5, 0, 'Desplazamiento'),
 Text(0.5, 0, 'Velocidad'),
 Text(0.5, 0, 'Tiempo')]
```



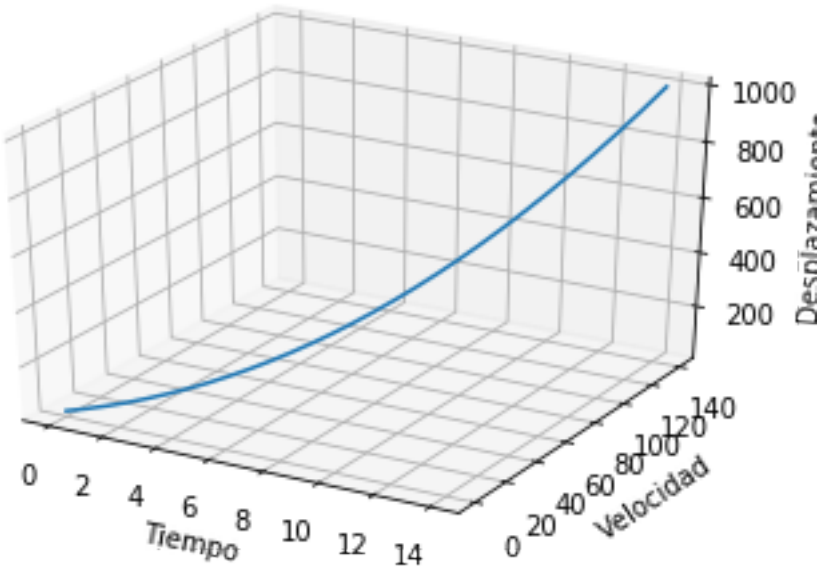
xi < 0, vxi = 0

[21]:
```
a = Vector(value=Coordenate(x=1, y=10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=random.randint(0, MAX_INT)))
v0 = Vector(value=Coordenate(x=0,y=0))


tf = 14
h = 0.01
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]

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)
    as_.append(a.y)


print("Posicion FInal\t\t:\t{}\nVelocidad Final\t:\t{}".format(yf.y, v0.y))
fig = plt.figure()
ax3d = plt.axes(projection='3d')
ax3d.plot(t, vy, dy)
ax3d.set(xlabel='Tiempo', ylabel='Velocidad', zlabel='Desplazamiento')
```

```
Tiempo inicial          :       0
Punto Inicial           :       25
Velocidad Inicial       :       0
Tiempo Final            :       14
Posicion FInal          :       1005.6999999999897
Velocidad Final :       139.99999999999633
```

[21]:
```
[Text(0.5, 0, 'Desplazamiento'),
 Text(0.5, 0, 'Velocidad'),
```

```
Text(0.5, 0, 'Tiempo')]
```



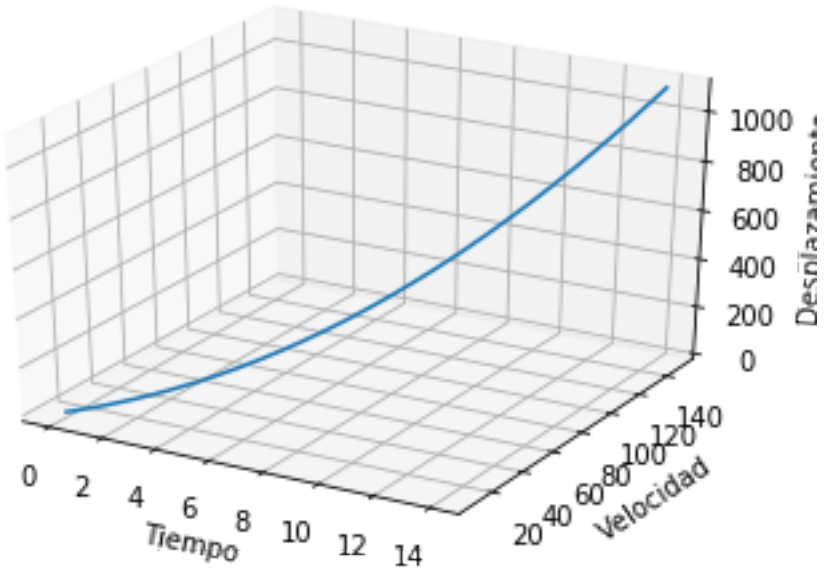
xi = 0, vxi > 0

[22]:
```
a = Vector(value=Coordenate(x=1, y=10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=0))
v0 = Vector(value=Coordenate(x=0,y=random.randint(0, MAX_INT)))


tf = 14
h = 0.01
yf = y0

print("Tiempo inicial\t\t:\t{}\nPunto Inicial\t\t:\t{}\nVelocidad Inicial\t:
 ↪\t{}\nTiempo Final\t\t:\t{}".format(t0, y0.y, v0.y, tf))


t = [ 0 ]
vy = [ v0.y ]
dy = [ y0.y ]
as_ = [ a.y ]

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)
    as_.append(a.y)


print("Posicion FInal\t\t:\t{}\nVelocidad Inicial\t:\t{}".format(yf.y, v0.y))
fig = plt.figure()
ax3d = plt.axes(projection='3d')
ax3d.plot(t, vy, dy)
ax3d.set(xlabel='Tiempo', ylabel='Velocidad', zlabel='Desplazamiento')
```

```
Tiempo inicial          :       0
Punto Inicial           :       0
Velocidad Inicial       :       9
Tiempo Final            :       14
Posicion FInal          :       1106.6999999999862
Velocidad Inicial       :       148.99999999999582
```

[22]:
```
[Text(0.5, 0, 'Desplazamiento'),
 Text(0.5, 0, 'Velocidad'),
 Text(0.5, 0, 'Tiempo')]
```

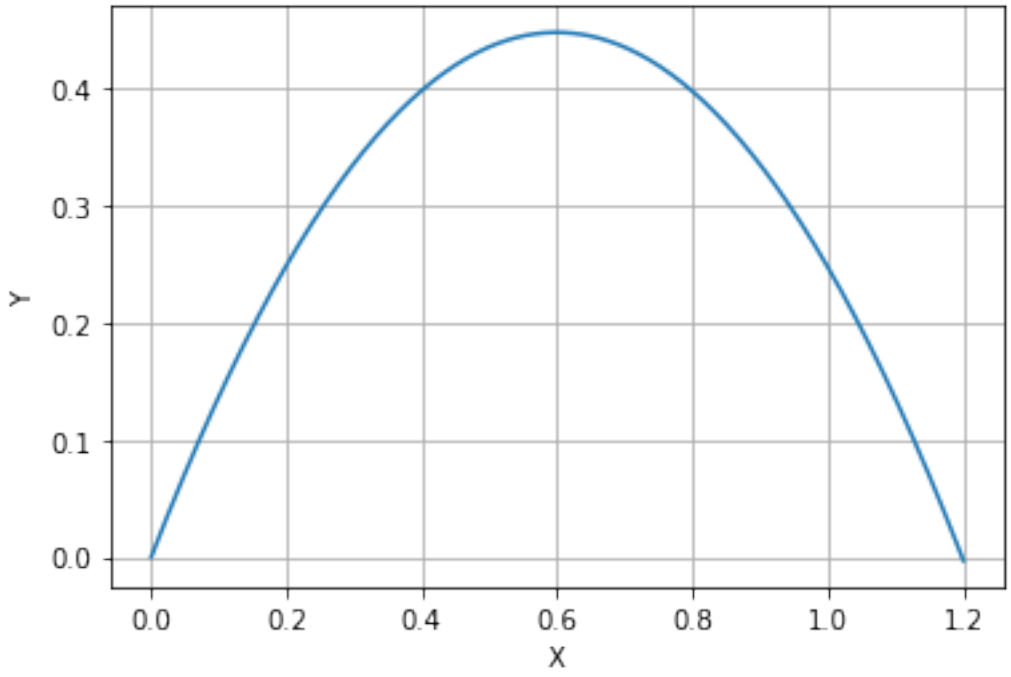


5.2 Movimiento en 2D y 3D con aceleración constante en la dirección y

5.2.1 Una partícula parte del reposo r = 0 m con una velocidad inicial vi = (2i + 3j) m/s. grafique su trayectoria hasta que la partícula cruce el eje x. De un ejemplo real.

[23]:
```
a = Vector(value=Coordenate(x=0, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=0))
v0 = Vector(value=Coordenate(x=2,y=3))


tf = 0.5
h = 0.001

yf = y0


xs = [ y0.x ]
ys = [ y0.y ]

while yf.y >= 0:
    v0 = v0 + a.mulEscalar(h)
    yf = yf + v0.mulEscalar(h)
    ys.append(yf.y)
    xs.append(yf.x)

print("Posicion Final\t\t:\t({}, {})\nVelocidad Final\t\t:\t({},{})".format(yf.
 ↪x,yf.y, v0.x, v0.y))
fig = plt.figure()
ax = plt.axes()
ax.plot(xs, ys)
ax.set(xlabel='X', ylabel='Y')
ax.grid()
```

```
Posicion Final          :       (1.2000000000000008, -0.0029999999999882157)
Velocidad Final         :       (2.0,-2.9999999999999605)
```

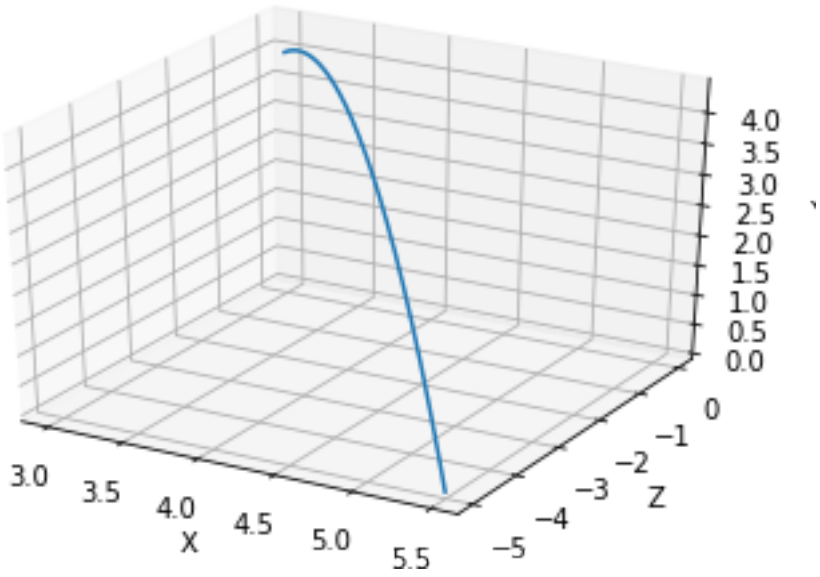


5.2.2 Una partícula parte del reposo r = (3i + 4k) m con una velocidad inicial v = (2i + 3j − 4k) m/s. Dibuje la trayectoria hasta que la partícula cruce el plano xz.

[24]:
```
a = Vector(value=Coordenate(x=0, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=3,y=4))
v0 = Vector(value=Coordenate(x=2,y=3, z = -4))


tf = 0.5
h = 0.001

yf = y0


xs = [ y0.x ]
ys = [ y0.y ]
zs = [ y0.z ]
while yf.y >= 0:
    v0 = v0 + a.mulEscalar(h)
    yf = yf + v0.mulEscalar(h)
    ys.append(yf.y)
```

```
    xs.append(yf.x)
    zs.append(yf.z)

print("Posicion Final\t\t:\t{}\nVelocidad Final\t\t:\t{}".format(yf.y, v0.y))
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot(xs, zs, ys)
ax.set(xlabel='X', ylabel='Z', zlabel='Y')
ax.grid()
```

```
Posicion Final          :       -0.002459999999920963
Velocidad Final         :       -9.429999999999824
```



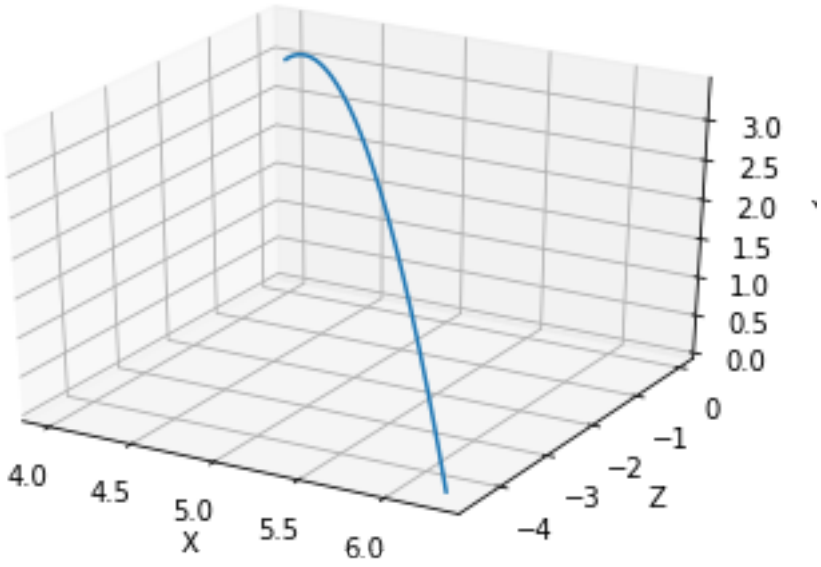
5.2.3 Una pelota se lanza desde el techo r = (4i + 3j) m con una velocidad inicial vi = (2i + 3j − 4k)m/s.

(a) Dibuje la trayectoria hasta que la partícula llegue al suelo.
(b) Determine el tiempo en que la pelota llega a una altura máxima
(c) En ese tiempo ubique las coordenadas de la pelota.
(d) Determine el alcance vectorialmente.

[25]:
```
a = Vector(value=Coordenate(x=0, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=4,y=3))
v0 = Vector(value=Coordenate(x=2,y=3, z = -4))
```

```
tf = 0.5
h = 0.0009

yf = y0
lv = v0

xs = [ y0.x ]
ys = [ y0.y ]
zs = [ y0.z ]
while yf.y >= 0:
    v0 = v0 + a.mulEscalar(h)
    yf = yf + v0.mulEscalar(h)
    t0 = t0 + h

    ys.append(yf.y)
    xs.append(yf.x)
    zs.append(yf.z)

    if v0.y == 0 or getSigno(v0.y)!=getSigno(lv.y):
        print ("Tiempo de la cima\t:\t{:.2f}\nCima en\t\t\t:\t({:.2f}, {:.2f},␣
↪{:.2f})".format(t0,yf.x,yf.y,yf.z))
    lv = v0



print("Alcance\t\t\t:\t({:.2f}, {:.2f}, {:.2f})".format(yf.x, yf.y,yf.z))
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot(xs, zs, ys)
ax.set(xlabel='X', ylabel='Z', zlabel='Y')
ax.grid()
```

```
Tiempo de la cima       :       0.30
Cima en                 :       (4.60, 3.45, -1.20)
Alcance                 :       (6.26, -0.00, -4.52)
```



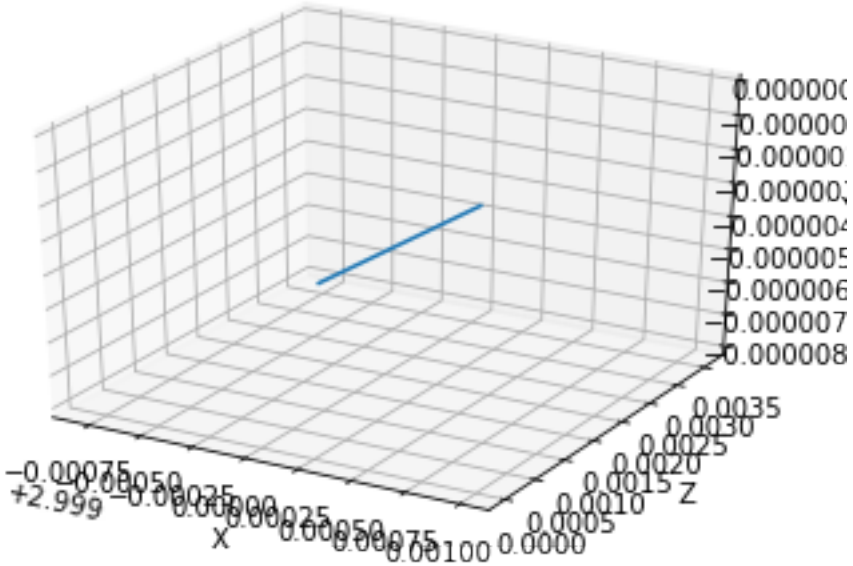
5.3 Movimiento en 2D y 3D con aceleración constante en dos direcciones

5.3.1 Una partícula parte del reposo desde r = (3i) m con una velocidad v = −2i m/s. Hay una ráfaga de viento cuya velocidad es vv = (4k) m/s. Determine la trayectoria de la partícula cuando llegue al suelo

[26]:
```
a = Vector(value=Coordenate(x=0, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=3))
v0 = Vector(value=Coordenate(x=-2))    +  Vector(value=Coordenate(z = 4))


tf = 0.5
h = 0.0009

yf = y0
lv = v0

xs = [ y0.x ]
ys = [ y0.y ]
zs = [ y0.z ]
while yf.y >= 0:
    v0 = v0 + a.mulEscalar(h)
    yf = yf + v0.mulEscalar(h)
    t0 = t0 + h
```

```
    ys.append(yf.y)
    xs.append(yf.x)
    zs.append(yf.z)



print("Alcance\t\t\t:\t({:.2f}, {:.2f}, {:.2f})".format(yf.x, yf.y,yf.z))
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot(xs, zs, ys)
ax.set(xlabel='X', ylabel='Z', zlabel='Y')
ax.grid()
```

```
Alcance                 :       (3.00, -0.00, 0.00)
```

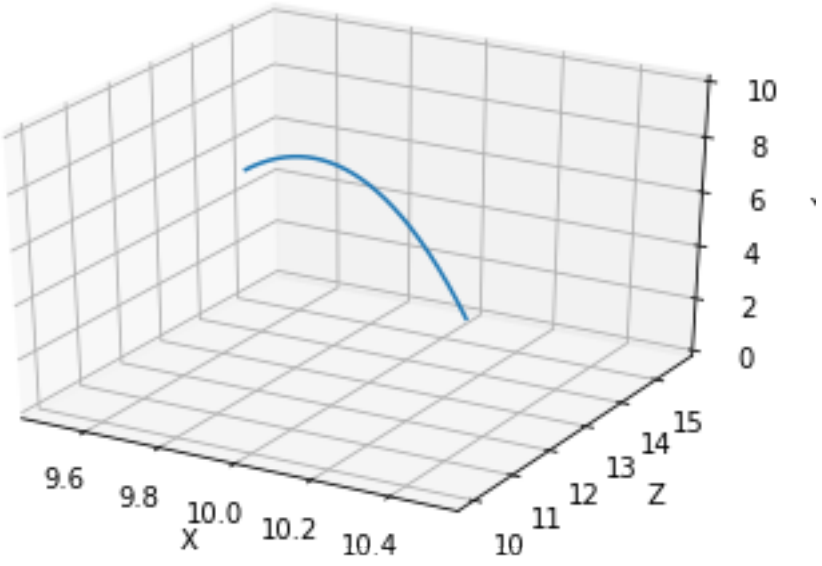


5.3.2 Una pelota se ubica en el techo de un edificio cuyas ubicación es r = (10i+ 10j + 10k)m. Luego un niño patea dicha pelota con la velocidad vp = 2i m/s. Pero se encuentra con la sorpresa que hay un viento cuya velocidad es vw = (−2i + 4k) m/s. Determine la trayectoria de la pelota hasta que llegue al suelo.

[27]:
```
a = Vector(value=Coordenate(x=0, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=10,y=10, z = 10))
v0 = Vector(value=Coordenate(x=2))   +  Vector(value=Coordenate(x = -2, z = 4))
```

```
tf = 0.5
h = 0.001

yf = y0
lv = v0

xs = [ y0.x ]
ys = [ y0.y ]
zs = [ y0.z ]
while yf.y >= 0:
    v0 = v0 + a.mulEscalar(h)
    yf = yf + v0.mulEscalar(h)
    t0 = t0 + h

    ys.append(yf.y)
    xs.append(yf.x)
    zs.append(yf.z)



print("Alcance\t\t\t:\t({:.2f}, {:.2f}, {:.2f})".format(yf.x, yf.y,yf.z))
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot(xs, zs, ys)
ax.set(xlabel='X', ylabel='Z', zlabel='Y')
ax.grid()
```

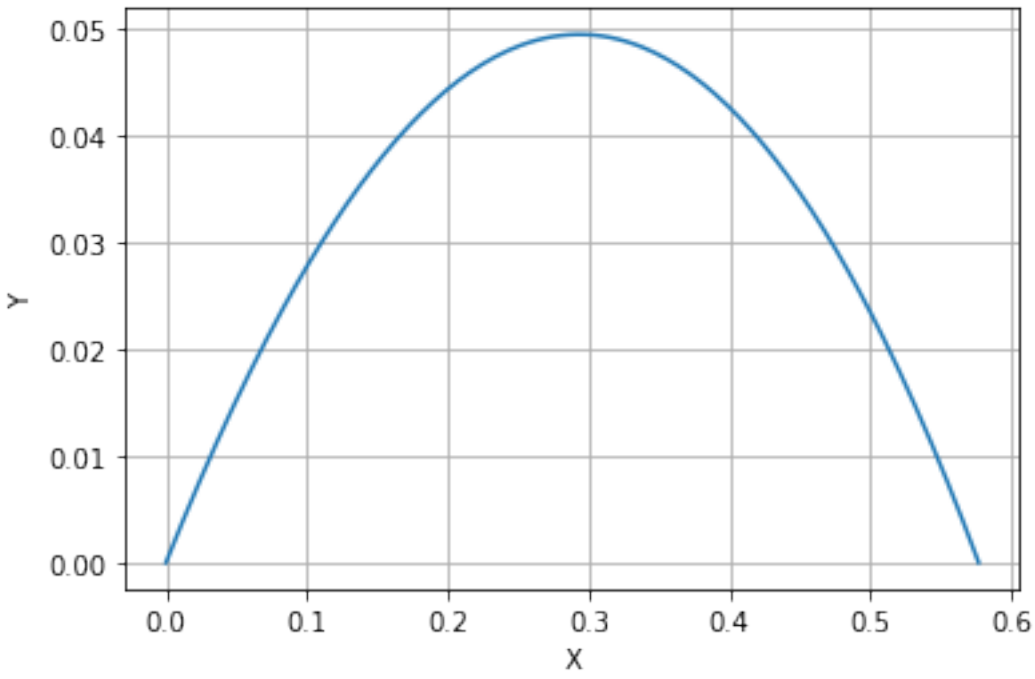
```
Alcance                 :       (10.00, -0.00, 15.66)
```



5.3.3 La ecuaciones de la cinemática de un movimiento parabólico cuando una partícula parte de (0, 0) son x = vxit, y y = vyit + gt2/2 (vxi = 1 m/s, vyi = 1 m/s y g = −10 m/s2) . Se presenta una ráfaga de viento cuyas ecuación es xw = vwit + awt 2/2 (vwi = 2 m/s y aw = −1 m/s2). Utilice lapiz y papel para encontrar la ecuación de la trayectoria

[28]:
```
a0 = Vector(value=Coordenate(x=0, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=0))
v0 = Vector(value=Coordenate(x=1, y=1))

vw = Vector(value=Coordenate(x=2))
aw = Vector(value=Coordenate(x=-1))

v = v0 + vw
a = a0 + aw

tf = 0.5
h = 0.001

yf = y0
lv = v0

xs = [ y0.x ]
ys = [ y0.y ]
while yf.y >= 0:
    v = v + a.mulEscalar(h)
    yf = yf  + v.mulEscalar(h)
    t0 = t0 + h
    ys.append(yf.y)
    xs.append(yf.x)



print("Alcance\t\t\t:\t({:.2f}, {:.2f})".format(yf.x, yf.y))
fig = plt.figure()
ax = plt.axes()
ax.plot(xs,ys)
ax.set(xlabel='X', ylabel='Y')
ax.grid()
```

```
Alcance                 :       (0.58, -0.00)
```



5.3.4 Un jugador de Golf golpea una bola con el driver ubicado en r = 0. La bola sale con velocidad inicial v = (5i+2j) m/s. Pero resulta que esta presente una ráfaga de viento con una aceleración aw. Dibuje la trayectoria de la bola hasta que llegue al suelo y dibuje la posición final, cuando

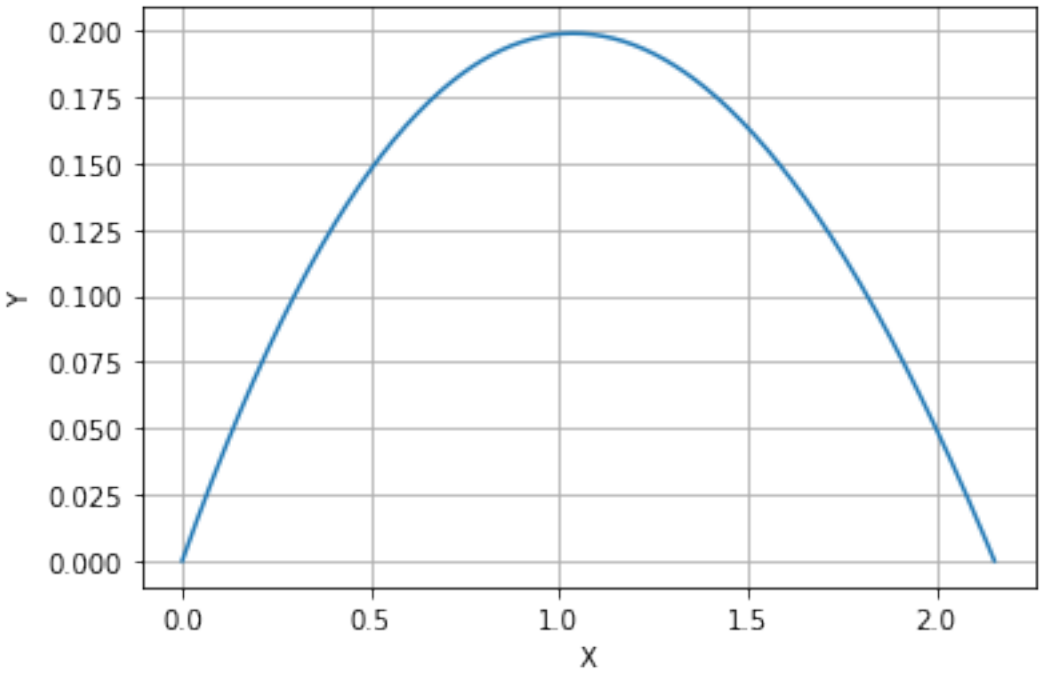
[29]:
```
a0 = Vector(value=Coordenate(x=0, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=0))
v0 = Vector(value=Coordenate(x=5, y=2))
```

(a) aw = +2i m/s2, con vwi = 0

[30]:
```
vw = Vector(value=Coordenate(x=0))
aw = Vector(value=Coordenate(x=2))

v = v0 + vw
a = a0 + aw

h = 0.001

yf = y0

xs = [ y0.x ]
```

```
ys = [ y0.y ]
while yf.y >= 0:
    v = v + a.mulEscalar(h)
    yf = yf  + v.mulEscalar(h)
    t0 = t0 + h
    ys.append(yf.y)
    xs.append(yf.x)



print("Alcance\t\t\t:\t({:.2f}, {:.2f})".format(yf.x, yf.y))
fig = plt.figure()
ax = plt.axes()
ax.plot(xs,ys)
ax.set(xlabel='X', ylabel='Y')
ax.grid()
```

```
Alcance                 :       (2.15, -0.00)
```

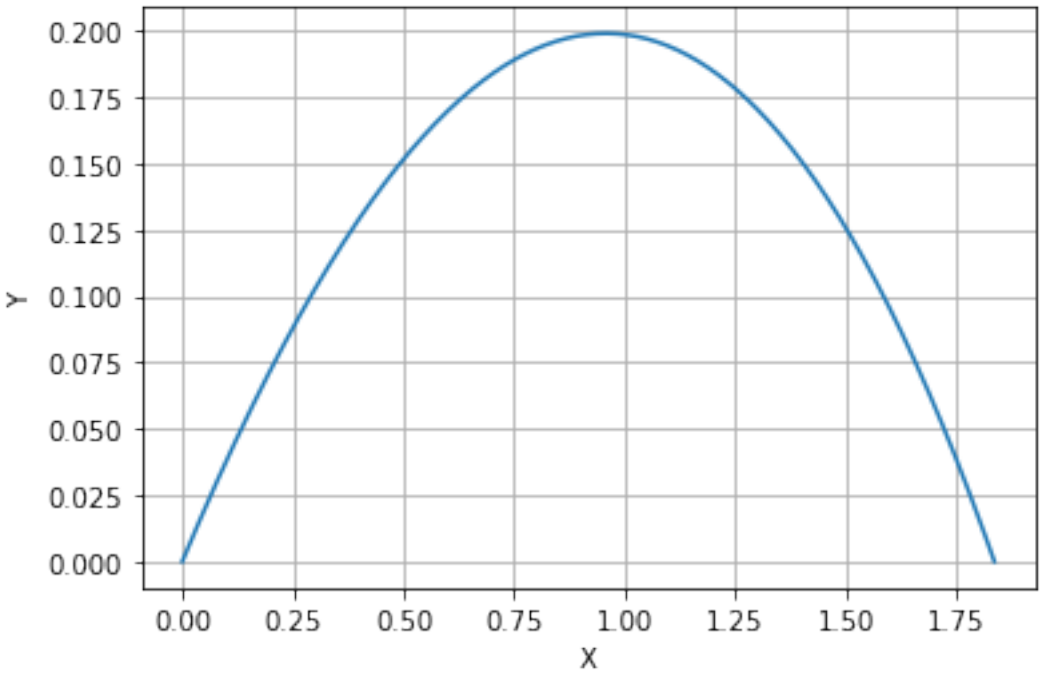


(b) aw = −2i m/s2, con vwi = 0

[31]:
```
vw = Vector(value=Coordenate(x=0))
aw = Vector(value=Coordenate(x=-2))

v = v0 + vw
```

```
a = a0 + aw

h = 0.001

yf = y0

xs = [ y0.x ]
ys = [ y0.y ]
while yf.y >= 0:
    v = v + a.mulEscalar(h)
    yf = yf  + v.mulEscalar(h)
    t0 = t0 + h
    ys.append(yf.y)
    xs.append(yf.x)



print("Alcance\t\t\t:\t({:.2f}, {:.2f})".format(yf.x, yf.y))
fig = plt.figure()
ax = plt.axes()
ax.plot(xs,ys)
ax.set(xlabel='X', ylabel='Y')
ax.grid()
```

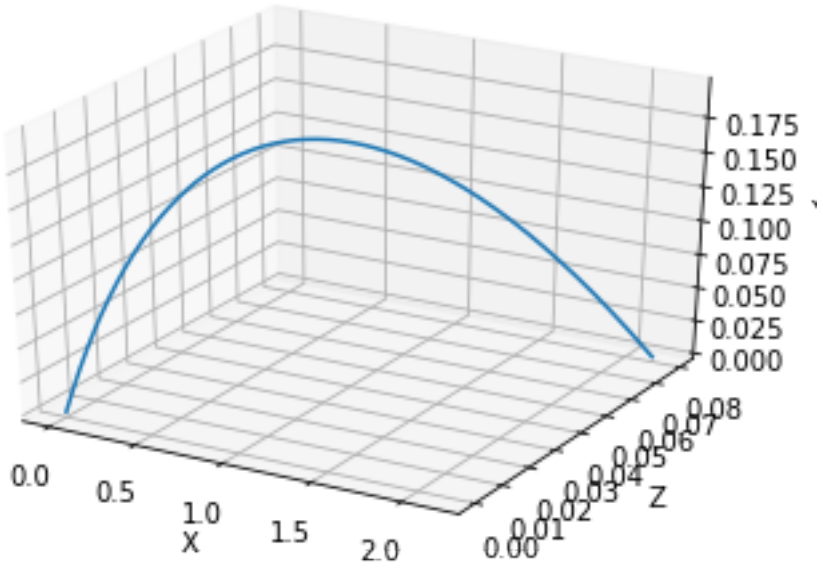
```
Alcance                 :       (1.84, -0.00)
```



(c) aw = (+2i + k) m/s2, con vwi = 0

[32]:
```
a0 = Vector(value=Coordenate(x=0, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=0))
v0 = Vector(value=Coordenate(x=5, y=2))

vw = Vector(value=Coordenate(x=0))
aw = Vector(value=Coordenate(x=2, z=1))

v = v0 + vw
a = a0 + aw

h = 0.001

yf = y0

xs = [ y0.x ]
ys = [ y0.y ]
zs = [ y0.z ]
while yf.y >= 0:
    v = v + a.mulEscalar(h)
    yf = yf  + v.mulEscalar(h)
    t0 = t0 + h
    ys.append(yf.y)
    xs.append(yf.x)
    zs.append(yf.z)



print("Alcance\t\t\t:\t({:.2f}, {:.2f}, {:.2f})".format(yf.x, yf.y, yf.z))
fig = plt.figure()
ax = plt.axes(projection = '3d')
ax.plot(xs,zs,ys)
ax.set(xlabel='X', ylabel = 'Z', zlabel='Y')
ax.grid()
```

```
Alcance                 :       (2.15, -0.00, 0.08)
```

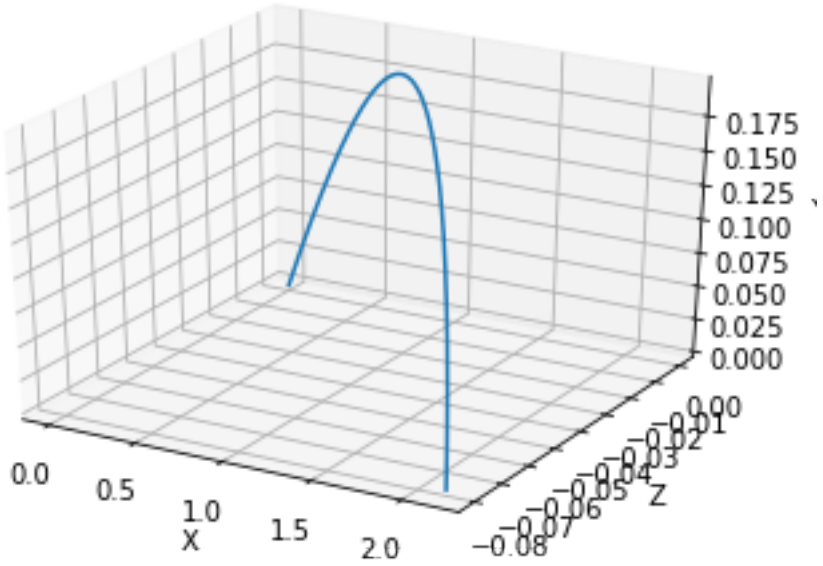


(d) aw = (+2i − k) m/s2, con vwi = 0

[33]:
```
a0 = Vector(value=Coordenate(x=0, y=-10))

t0 = 0
y0 = Vector(value=Coordenate(x=0,y=0))
v0 = Vector(value=Coordenate(x=5, y=2))


vw = Vector(value=Coordenate(x=0))
aw = Vector(value=Coordenate(x=2, z=-1))

v = v0 + vw
a = a0 + aw

h = 0.001

yf = y0

xs = [ y0.x ]
ys = [ y0.y ]
zs = [ y0.z ]
while yf.y >= 0:
    v = v + a.mulEscalar(h)
    yf = yf  + v.mulEscalar(h)
    t0 = t0 + h
    ys.append(yf.y)
```

```
    xs.append(yf.x)
    zs.append(yf.z)



print("Alcance\t\t\t:\t({:.2f}, {:.2f}, {:.2f})".format(yf.x, yf.y, yf.z))
fig = plt.figure()
ax = plt.axes(projection = '3d')
ax.plot(xs,zs,ys)
ax.set(xlabel='X', ylabel = 'Z', zlabel='Y')
ax.grid()
```

```
Alcance                 :       (2.15, -0.00, -0.08)
```



[ ]:

5.4 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()
```

[ ]:

5.5 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__)
```

5.6 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
```

[ ]: