Day 27

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# Runge-Kutta 4nd order

def f(r,t):
    y = r[0]
    v = r[1]
    fy = v
    fv = -9.81
    return(np.array([fy,fv],float))

# define boundary conditions

t0 = 0.0 # starting point
tf = 10.0 # ending point
N = 1000 # number of points between a and b
dt = (tf-t0)/N

r = np.array([100,30],float) # initial condition

tpoints = np.arange(t0, tf, dt)
ypoints = []
vpoints = []

for t in tpoints:
    ypoints.append(r[0])
    vpoints.append(r[1])
    k1 = dt*f(r,t)
    k2 = dt*f(r+0.5*k1,t+0.5*dt)
    k3 = dt*f(r+0.5*k2,t+0.5*dt)
    k4 = dt*f(r+k3, t+dt)
    r = r + (k1+2*k2+2*k3+k4)/6
```

In [3]:
```
fig0, ax0 = plt.subplots()
ax0.plot(tpoints, ypoints)
ax0.plot(tpoints, vpoints)
```

Out[3]: [<matplotlib.lines.Line2D at 0x7fef4bda0470>]

what about linear drag?

In [4]:
```
# Runge-Kutta 4nd order

def f(r,t):
    y = r[0]
    v = r[1]
    fy = v
    c = 1
    fv = -9.81 - c*v
    return(np.array([fy,fv],float))

# define boundary conditions

t0 = 0.0 # starting point
tf = 5.0 # ending point
N = 1000 # number of points between a and b
dt = (tf-t0)/N

r = np.array([100,0],float) # initial condition

tpoints = np.arange(t0, tf, dt)
ypoints = []
vpoints = []

for t in tpoints:
    ypoints.append(r[0])
    vpoints.append(r[1])
    k1 = dt*f(r,t)
    k2 = dt*f(r+0.5*k1,t+0.5*dt)
    k3 = dt*f(r+0.5*k2,t+0.5*dt)
    k4 = dt*f(r+k3, t+dt)
    r = r + (k1+2*k2+2*k3+k4)/6
```

In [5]:

```
fig1, ax1 = plt.subplots()
ax1.plot(tpoints, ypoints)
```

Out[5]: [<matplotlib.lines.Line2D at 0x7fef4afbd518>]

In [6]:

```
fig2, ax2 = plt.subplots()
ax2.plot(tpoints, vpoints)
```

Out[6]: [<matplotlib.lines.Line2D at 0x7fef4b104fd0>]

What about quadratic drag??

In [7]:

```
# Runge-Kutta 4nd order

def f(r,t):
    y = r[0]
    v = r[1]
    fy = v
    c = .001
    fv = -9.81 - c*v**2
    return(np.array([fy,fv],float))

# define boundary conditions

t0 = 0.0 # starting point
tf = 5.0 # ending point
N = 1000 # number of points between a and b
dt = (tf-t0)/N

r = np.array([100,0],float) # initial condition

tpoints = np.arange(t0, tf, dt)
ypoints = []
vpoints = []

for t in tpoints:
    ypoints.append(r[0])
    vpoints.append(r[1])
    k1 = dt*f(r,t)
    k2 = dt*f(r+0.5*k1,t+0.5*dt)
    k3 = dt*f(r+0.5*k2,t+0.5*dt)
    k4 = dt*f(r+k3, t+dt)
    r = r + (k1+2*k2+2*k3+k4)/6
```

In [8]:

```
fig3, ax3 = plt.subplots()
ax3.plot(tpoints, ypoints)
```

Out[8]: [<matplotlib.lines.Line2D at 0x7fef4b0ae7f0>]

In [9]:

```
# Runge-Kutta 4nd order

def f(r,t):
    y = r[0]
    v = r[1]
    fy = v
    c = 1
    fv = -9.81 - c*v
    return(np.array([fy,fv],float))

def cRK4(f, tf, x0, v0, t0=0, dt=2**-5):

    r = np.array([x0,v0],float) # initial condition

    tpoints = np.arange(t0, tf, dt)
    xpoints = []
    vpoints = []

    for t in tpoints:
        xpoints.append(r[0])
        vpoints.append(r[1])
```

```python
        k1 = dt*f(r,t)
        k2 = dt*f(r+0.5*k1,t+0.5*dt)
        k3 = dt*f(r+0.5*k2,t+0.5*dt)
        k4 = dt*f(r+k3, t+dt)
        r = r + (k1+2*k2+2*k3+k4)/6

    return(tpoints, xpoints, vpoints)
```

In [10]:

```python
def func0(r,t):
    y = r[0]
    v = r[1]
    fy = v
    c = 1
    fv = -9.81 - c*v
    return(np.array([fy,fv],float))

t0, y0, v0 = cRK4(func0, 5, 100, 0)
```

In [11]:

```python
fig4, ax4 = plt.subplots()
ax4.plot(t0,v0)
```

Out[11]: [<matplotlib.lines.Line2D at 0x7fef4b031f60>]

In [26]:

```python
def func1(r,t):
    y = r[0]
    v = r[1]
    fy = v
    c1 = 25
    c2 = 20
    fv = -9.81 +c1 - c2*v
    return(np.array([fy,fv],float))

t1, y1, v1 = cRK4(func1, 2, -1, 0)
```

In [27]:

```python
fig5, ax5 = plt.subplots()
ax5.plot(t1,y1)
ax5.plot(t1,v1)
```

Out[27]: [<matplotlib.lines.Line2D at 0x7fef4ac46198>]

In [ ]: