Day 27

In [ ]:

```
```

In [10]:

```
# 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 [11]:

```
fig0, ax0 = plt.subplots()
ax0.plot(tpoints, ypoints)
ax0.plot(tpoints, vpoints)
```

Out[11]:

```
[<matplotlib.lines.Line2D at 0x7f06a52a3ac8>]
```

what about linear drag?

In [20]:

```
# 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 [21]:
```
fig1, ax1 = plt.subplots()
ax1.plot(tpoints, ypoints)
```

Out[21]:
```
[<matplotlib.lines.Line2D at 0x7f06a4fa4b38>]
```

In [22]:
```
fig2, ax2 = plt.subplots()
ax2.plot(tpoints, vpoints)
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x7f06a4f210f0>]
```

What about quadratic drag??

In [30]:
```
# 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 [32]:
```
fig3, ax3 = plt.subplots()
ax3.plot(tpoints, ypoints)
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x7f069b05e470>]
```

In [33]:
```
# 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])
        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 [34]:

```
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 [36]:

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

```
/home/eric/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: RuntimeWarning: More than 20 figures h
ave been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until exp
licitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max_open_warnin
g`).
  """Entry point for launching an IPython kernel.
```

Out[36]:

```
[<matplotlib.lines.Line2D at 0x7f069aa74470>]
```

In [ ]:

In [ ]:

In [ ]: