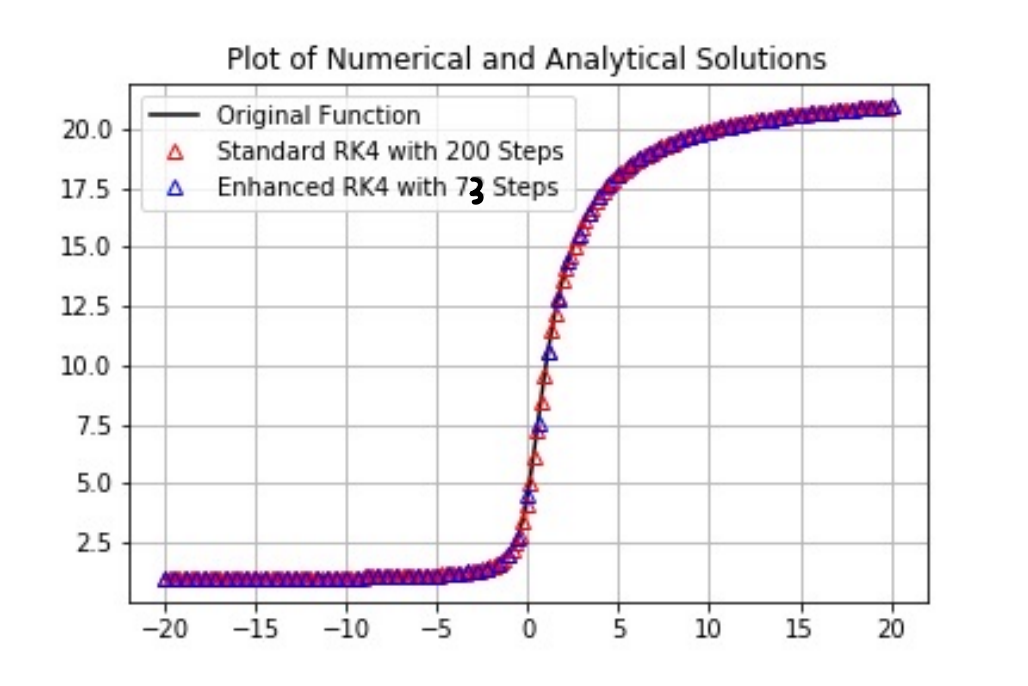
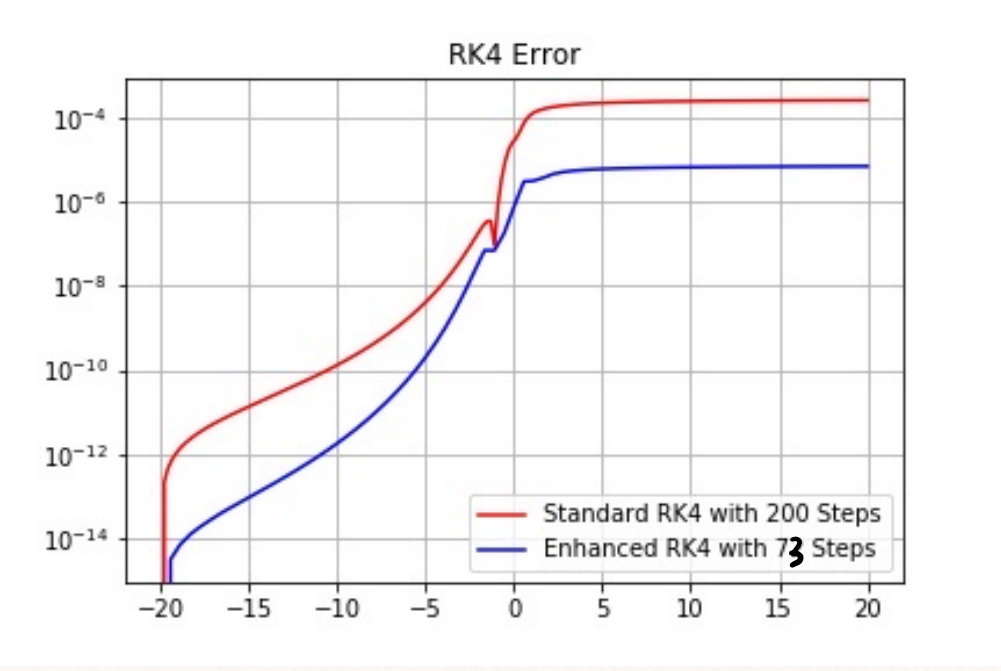
Prob 1

$$\frac{dy}{dx} = f(x,y) \xrightarrow{4^{th}\ Order} y(x+h) = y(x) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

[Plots and code below:]

$$N_{steps} = 200\left(\frac{4}{11}\right) \approx 73$$





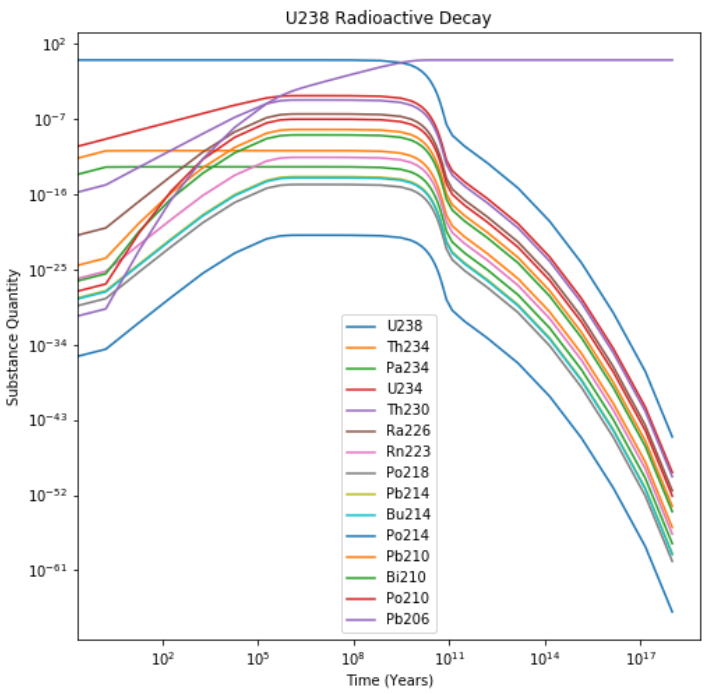

```
Standard RK4 RMS Error:  0.00012015197423066756
Enhanced RK4 RMS Error:  3.2111077299406217e-06
RK4 Enhancement is  37.417609228853046  times better on average...
```

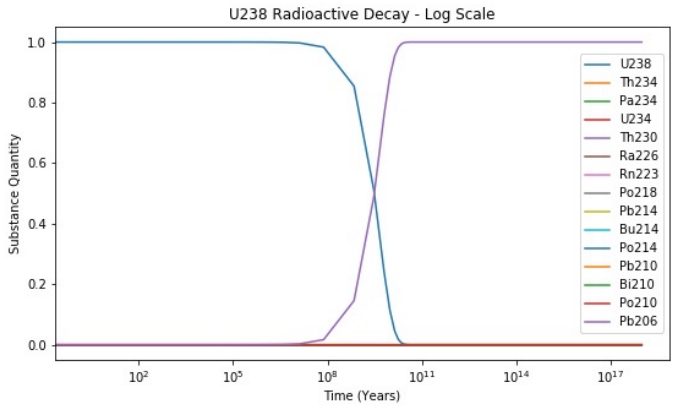
```python

import numpy as np
from matplotlib import pyplot as plt

#Standard RK4 Evalutation Below
def RationalExpression(x,y):
    dyBYdx = y/(1+x**2)
    return dyBYdx


def f(x,y):
    dyBYdx = np.asarray([y[1],-y[0]])
    return dyBYdx

def rk4_step(fun,x,y,h):
    k1 = h * fun(x,y)
    k2 = h * fun(x+h/2,y+k1/2)
    k3 = h * fun(x+h/2,y+k2/2)
    k4 = h * fun(x+h,y+k3)
    dy = (k1 + 2*k2 + 2*k3 + k4)/6
    return y + dy

#Stepper: takes a step length h, compares to h/2, cancels out leading order error from RK4

def rk4_stepd(fun,x,y,h):
    y1 = rk4_step(fun, x, y, h)
    y2a = rk4_step(fun, x, y, h/2)
    y2b = rk4_step(fun, x+h/2, y2a, h/2)
    return y2b + (y2b - y1)/15


# Let's Integrate using 200 Steps as requested in Question

y0 = 1
xSpace1 = np.linspace(-20, 20, 200)
h = np.median(np.diff(xSpace1))
ySpace1 = np.zeros(len(xSpace1))
ySpace1[0]=y0

for i in range(len(xSpace1)-1):
    ySpace1[i+1]=rk4_step(RationalExpression, xSpace1[i], ySpace1[i], h)
```

```python
# Let's Integrate using 200 Steps as requested in Question

y0 = 1
xSpace1 = np.linspace(-20, 20, 200)
h = np.median(np.diff(xSpace1))
ySpace1 = np.zeros(len(xSpace1))
ySpace1[0]=y0

for i in range(len(xSpace1)-1):
    ySpace1[i+1]=rk4_step(RationalExpression, xSpace1[i], ySpace1[i], h)

#stepd should use roughly 4/11 times as many steps to evaluate the same number of function evaluations as the original RK4 method

y0 = 1
xSpace2 = np.linspace(-20, 20, 73) #Should be roughly 72.7 steps, rounded up
h = np.median(np.diff(xSpace2))
ySpace2 = np.zeros(len(xSpace2))
ySpace2[0] = y0

for i in range(len(xSpace2)-1):
    ySpace2[i+1]=rk4_stepd(RationalExpression, xSpace2[i], ySpace2[i], h)

# Compare error to real functions
yAnalytical1 = np.e**(np.arctan(xSpace1) + np.arctan(20))
yAnalytical2 = np.e**(np.arctan(xSpace2) + np.arctan(20))

yError1 = np.abs(yAnalytical1 - ySpace1)
yError2 = np.abs(yAnalytical2 - ySpace2)

STND = 'Standard RK4 with 200 Steps'
ENH = 'Enhanced RK4 with 73 Steps'

# Numerical and Analytical Solutions Plot

plt.figure()
plt.title('Plot of Numerical and Analytical Solutions')
plt.plot(xSpace1,ySpace1,'k', label = 'Original Function')
plt.plot(xSpace1,ySpace1,'r^', markerfacecolor = 'none', label = STND)
plt.plot(xSpace2,ySpace2,'b^', markerfacecolor = 'none', label = ENH)
plt.legend()
plt.grid()
plt.savefig('SolutionsP3Q1.png')
plt.show()
```

```python

# Error Comparison

print('Standard RK4 RMS Error: ', np.std(yError1))
print('Enhanced RK4 RMS Error: ', np.std(yError2))
print('RK4 Enhancement is ', np.std(yError1)/np.std(yError2), ' times better on average...')
```

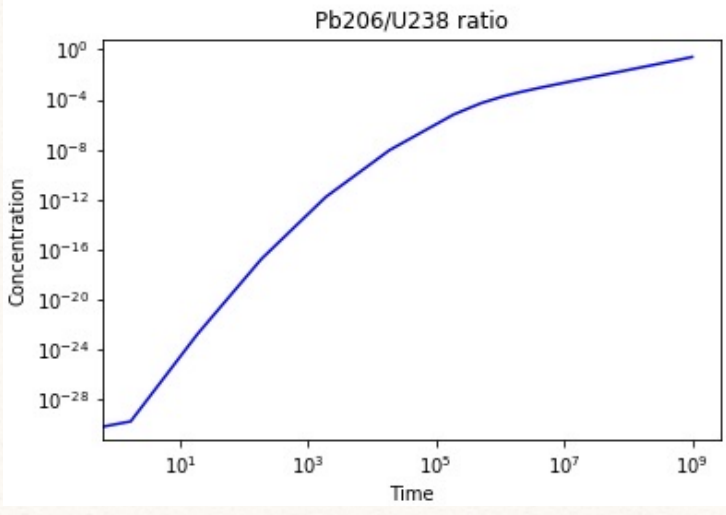
Problem 2



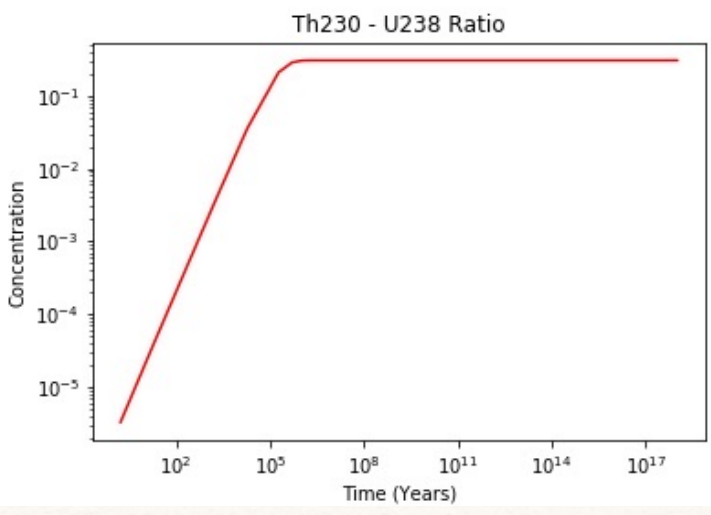


↑ Clearly demonstrates relationship between U238 and Pb206...

↑ Allows us to see how other non-crucial elements decay with time.



Region of Interest...



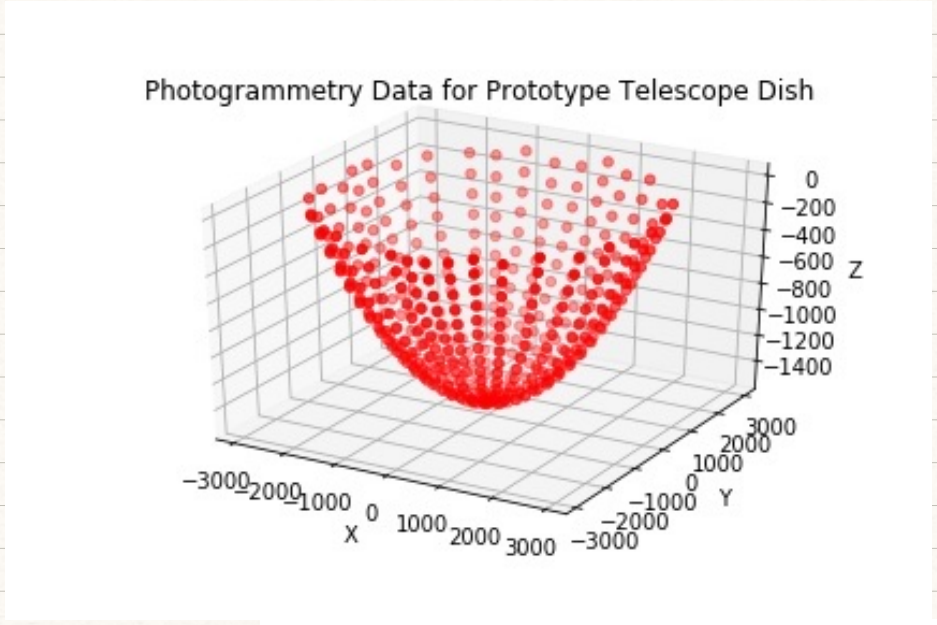
Ratio reaches point of equilibrium...

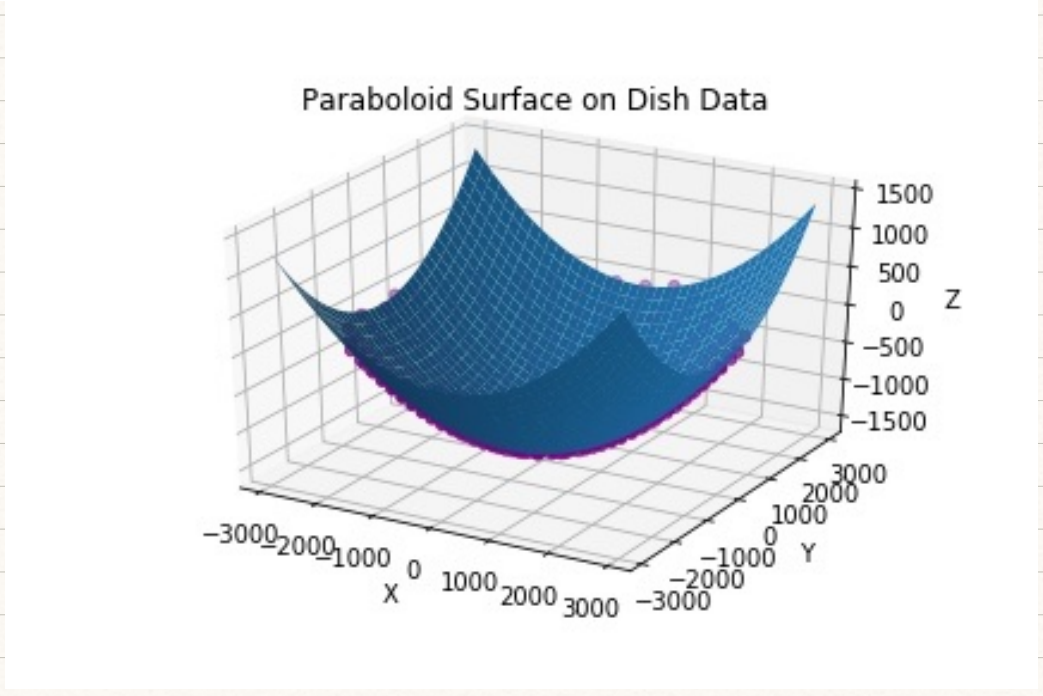
```

import numpy as np
from matplotlib import pyplot as plt

#Standard RK4 Evalutation Below
def RationalExpression(x,y):
    dyBYdx = y/(1+x**2)
    return dyBYdx


def f(x,y):
    dyBYdx = np.asarray([y[1],-y[0]])
    return dyBYdx

def rk4_step(fun,x,y,h):
    k1 = h * fun(x,y)
    k2 = h * fun(x+h/2,y+k1/2)
    k3 = h * fun(x+h/2,y+k2/2)
    k4 = h * fun(x+h,y+k3)
    dy = (k1 + 2*k2 + 2*k3 + k4)/6
    return y + dy

#Stepper: takes a step length h, compares to h/2, cancels out leading order error from RK4

def rk4_stepd(fun,x,y,h):
    y1 = rk4_step(fun, x, y, h)
    y2a = rk4_step(fun, x, y, h/2)
    y2b = rk4_step(fun, x+h/2, y2a, h/2)
    return y2b + (y2b - y1)/15


# Let's Integrate using 200 Steps as requested in Question

y0 = 1
xSpace1 = np.linspace(-20, 20, 200)
h = np.median(np.diff(xSpace1))
ySpace1 = np.zeros(len(xSpace1))
ySpace1[0]=y0

for i in range(len(xSpace1)-1):
    ySpace1[i+1]=rk4_step(RationalExpression, xSpace1[i], ySpace1[i], h)

#stepd should use roughly 4/11 times as many steps to evaluate the same number of function evaluations as the original RK4 method

y0 = 1
xSpace2 = np.linspace(-20, 20, 73) #Should be roughly 72.7 steps, rounded up
h = np.median(np.diff(xSpace2))
ySpace2 = np.zeros(len(xSpace2))
ySpace2[0] = y0

for i in range(len(xSpace2)-1):
    ySpace2[i+1]=rk4_stepd(RationalExpression, xSpace2[i], ySpace2[i], h)

# Compare error to real functions
yAnalytical1 = np.e**(np.arctan(xSpace1) + np.arctan(20))
yAnalytical2 = np.e**(np.arctan(xSpace2) + np.arctan(20))

yError1 = np.abs(yAnalytical1 - ySpace1)
yError2 = np.abs(yAnalytical2 - ySpace2)

STND = 'Standard RK4 with 200 Steps'
ENH = 'Enhanced RK4 with 72 Steps'

# Numerical and Analytical Solutions Plot

plt.figure()
plt.title('Plot of Numerical and Analytical Solutions')
plt.plot(xSpace1,ySpace1,'k', label = 'Original Function')
plt.plot(xSpace1,ySpace1,'r^', markerfacecolor = 'none', label = STND)
plt.plot(xSpace2,ySpace2,'b^', markerfacecolor = 'none', label = ENH)
plt.legend()
plt.grid()
plt.savefig('SolutionsP3Q1.png')
plt.show()
```

Problem 3 :

$$z = a(x^2+y^2) - 2ax_0x - 2ayy_0 + (x_0^2a + ay_0^2 + z_0)$$

$$A(x^2+y^2) + Bx + Cy + D = z \quad \rightsquigarrow \quad \begin{cases} a = A \\ x_0 = -\frac{B}{2}A \\ y_0 = -\frac{C}{2A} \\ z_0 = -\frac{B^2+C^2-4AD}{4A} \end{cases}$$





↳ All code provided on github...