```python
"""Problem Set 1.

Observational Techniques
Jonas Powell
10/31/18
"""

import numpy as np
import matplotlib.pyplot as plt
from astropy.constants import G, M_earth, R_earth, c
r_earth = R_earth.value
c = c.value


# PART A

pos1 = [19.80159, 155.45581]
pos2 = [17.75652, 64.58376]

def find_distance(pos1, pos2, to_return='arc'):
    """Find the physical distance (in meters) between two points.

    Could use Lambert's Formula for Long Lines to recognize
    ellipsoidal Earth (not implemented)

    Args:
        pos1, pos2 (tuples): Lat, Long in decimal degrees.
    """
    phi1, lam1 = pos1
    phi2, lam2 = pos2

    phi1 *= np.pi/180
    phi2 *= np.pi/180
    lam1 *= np.pi/180
    lam2 *= np.pi/180
    d_lam = abs(lam2 - lam1)

    d_sig = np.arccos(np.sin(phi1) * np.sin(phi2) +
                      np.cos(phi1) * np.cos(phi2) * np.cos(d_lam))
    sig_error = 0.1 * np.pi/180

    error_angle, error_radius = 1e-5, 0
    error_coeff = np.sqrt((error_angle/d_sig)**2 + (error_radius/r_earth)**2)

    if to_return == 'cord':
```

```
        cord = 2 * r_earth * np.sin(d_sig/2)
        return [cord, cord * error_coeff]
    elif to_return == 'arc':
        distance = r_earth * d_sig
        return [distance, distance * error_coeff]
    elif to_return == 'angle':
        return [d_sig, sig_error]
    else:
        return "Invalid return request; choose 'arc', 'cord', or 'angle'."


find_distance(pos1, pos2, to_return='arc')
find_distance(pos1, pos2, to_return='cord')
find_distance(pos1, pos2, to_return='angle')


def find_angres_error(lam, sig_lam, d, sig_d):
    """Find the angular resolution and its error.

    Args:
        lam (float): Wavelength of observation, in meters.
        sig_lam (float): Variance of wavelength.
        d (float): Baseline length.
        sig_d (float): Variance of baseline length.
    """
    theta = lam/d
    sig_theta = theta * np.sqrt((sig_lam/lam)**2 + (sig_d/d)**2)
    return [theta, sig_theta]


d = find_distance(pos1, pos2, to_return='cord')[0]
sig_d = find_distance(pos1, pos2, to_return='cord')[1]
find_angres_error(0.1, 0, d, sig_d)


# PROBLEM 2
def get_relativistic_stuff(v_sat, h_sat):
    """Calculate the relativistic goodness.

    Args:
        v_sat (float): The satellite's angular velocity, in meters per seconds.
        h_sat (float): The satellite's altitude (height), in meters.
    """
    secs_to_years = 60*60*24*365

```

```python
    special_stuff = v_sat**2 / (2 * c**2) * secs_to_years
    general_stuff = G.value * M_earth.value * c**(-2) * \
        (1/(R_earth.value + h_sat) - 1/R_earth.value) * secs_to_years

    all_stuff = (special_stuff + general_stuff)

    print "Special Stuff:", special_stuff
    print "General Stuff:", general_stuff
    return all_stuff


h_sat = 540 * 1e3
period = 95 * 60 + 28
sat_circumference = 2 * np.pi * (r_earth + h_sat)
v_sat = sat_circumference/period
get_relativistic_stuff(v_sat, h_sat)


def get_final_time_delay(t=3600):
    """Calculate the time delay for light reaching our two observatories."""
    R = R_earth.value
    H = 5.4e5
    alt = 4205

    # Calculate the change in angular position in the course of an hour
    theta_MK = 2 * np.pi * t/86400
    theta_HST_dt = 2 * np.pi * t/5728
    """Rather than just d_theta/dt, we want its offset from theta=0
    (i.e. where Mauna Kea initially was), so correct for that:"""
    theta_HST = 2*np.pi - ((np.pi/2) +
                           np.arccos(R_earth.value/(R_earth.value + H)) +
                           theta_HST_dt)
    theta_i = theta_MK + theta_HST

    alpha = np.arctan((R + alt) *
                      np.sin(theta_i) / (R + H - (R + alt) * np.cos(theta_i)))
    theta = alpha + theta_HST - np.pi/2
    l = (R + H - (R + alt) * np.cos(theta_i))/(np.cos(alpha))

    d = l * np.sin(theta)
    dt = d/c
    return [d/R_earth.value, dt]


get_final_time_delay()
```

```python



def plot_delays():
    """Playing around with plotting the delays."""
    l_delays, t_delays = [], []
    ts = np.arange(10, 7000, 50)
    for dt in ts:
        l_delays.append(get_final_time_delay(t=dt)[0])
        t_delays.append(get_final_time_delay(t=dt)[1])

    fig, (ax1, ax2) = plt.subplots(1, 2)
    ax1.plot(ts, l_delays)
    ax2.plot(ts, t_delays)

    plt.show()



# The End
```