Galactic Astronomy: Problem Set 3

Jonas Powell, *Wesleyan University*

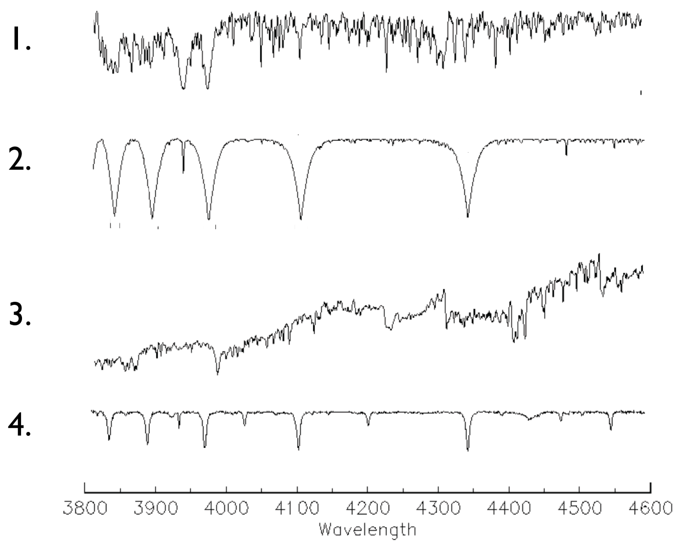
February 28, 2019

Due: Thursday, Feb. 28 by midnight. Late papers are not accepted. If you cannot complete the assignment, hand in what you have completed before the deadline. Consider the deadline to be like the boarding time for an airplane, or the deadline for a grant submission to NASA or NSF. If you miss the deadline, you do not get on the airplane, no matter how good your excuse is. If you miss an NSF or NASA deadline, you do not get the grant, no matter how good your project is. The best advice is ... finish early. You can submit multiple times, right up to the deadline. Whatever your latest submission is, when the deadline occurs, is what will be graded.

Problem 1. There is an error in the book on page 108? What correction is required?

Answer 1. On page 108, the author claims that, in the high-T regime, Planck's Law reduces to $B_\nu(T) = (2kT\nu^2)/(hc^2)$. This is not the case; the author has left an extra factor of h in the denominator. Instead, the Rayleigh-Jeans law says that $B_\nu(T) = 2kT\nu^2c^{-2}$.

Problem 2. Become familiar enough with stellar spectra that you can easily classify, at a glance, almost any spectrum into one of the main classes (OBAFGKM) to within plus or minus 1 class. Note that I am not talking about subclasses here, i.e telling a G2 star from a G3 star, which is difficult. I am talking about telling a G star from an A star or an M star. As an illustration of your prowess, how would you classify each of the stars shown below.



Answer 2. These are my classifications:

- Spectrum 1 is a K-star. We can tell this by its two distinctive absorption lines between 3900 and 4000Å, which are characteristic of K stars.
- Spectrum 2 is likely an A-star (or maybe a B), because of its generally clean, featureless spectrum save for large, clear lines.
- Spectrum 3 is an M-star. We can tell this because it does not have the major lines that Spectrum 1 had, but rather just a generally choppy surface that indicates an M-star.
- Spectrum 4 is an O-star. We know this because, like Spectrum 2, it is generally quite clean and free of minor features, but the significant lines that it does have are shallower than those of Spectrum 2, indicating that it is likely an O.

Problem 3. Make an HR diagram that compares a sample of the nearest stars (100 or so would do) with a sample of the brightest stars (chosen by apparent magnitude). The Sun, of course, will be in both data sets, since it is the nearest and the brightest. Plot everything on the same diagram and use different symbols and/or colors to differentiate between the nearest and the brightest.

There are a variety of catalogs you could use for this, including Gaia, Hipparcos, Gliese, Yale, etc. You may choose where to get your data and exactly what quantities you wish to plot. For example, luminosities could be in terms of M_V, M_{bol}, $L/L_\odot$, etc. and effective temperatures could be in terms of B-V, some other color index, T_e, etc. These are up to you! Make a great looking plot and then describe it and what it tells you about the stellar population in the Galaxy (and other galaxies, as it turns out).

Answer 3. First off: As I look back over this, I'm finding some weirdness in my Gaia query and am seeing that reflected here. However, I think the overarching message is still alright, but if I had more time, this is something I would go back and fix.

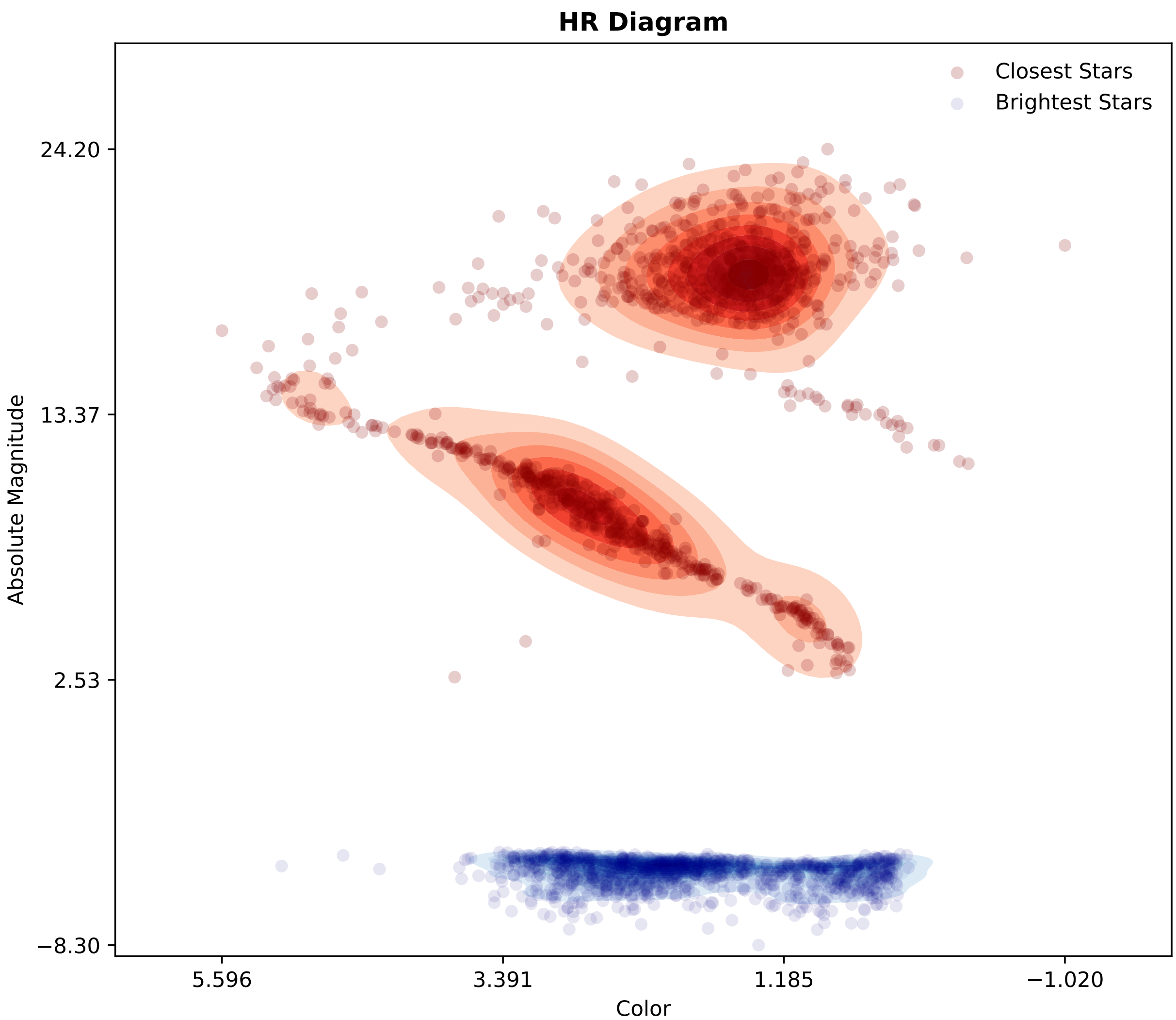


Figure 1: The HR Diagram, composed of the nearest thousand stars and the brightest thousand stars.

Given the enormity of Gaia and the value of large numbers, I chose to select the thousand brightest and thousand nearest stars. This yielded a more filled HR diagram, obviously, but visual inspection showed that it didn't change the qualitative structure of it much.

We immediately may see that the brightness cutoff we implemented, selecting only the top n brightest stars, is working by the fact that all those stars selected by brightness are clustered below some cutoff line on the y-axis. These stars tend to be clustered within a fairly non-extreme color range, from about 0 to 3.4.

The stars closest to us, of course, are much more evenly distributed across the HR diagram. Oddly, it seems like a significant portion of the stars are giants.

Problem 4. Determine the stellar number density and mass density in the solar neighborhood based on the sample of the 100 or so nearest stars that you used in problem 3. Note that you will need to assign masses to these objects, using a mass-luminosity relationship, which you adopt. Report your results in several different ways, so that you can get a feeling for what they mean. In particular, give the number density in units of stars per pc^{-3}, and the mass density in terms of M$_\odot$ pc^{-3}, gm cm^{-3}, and H-atoms cm^{-3}.

Answer 4. Using the same Gaia data that we used above, we may calculate varies densities in the local Universe. These numbers may be bad, reflecting the same errors I likely made above. Still, I think the logic behind my derivations is reasonable.

Densities

N_{stars} pc^{-3}	0.071
M$_\odot$ pc^{-3}	0.058
gm cm^{-3}	3.984×10^{-24}
N_H cm^{-3}	2379.87

How I found these values:

- N_{STARS} PC^{-3}: We may find a number density by dividing the total number of stars in the sample by a volume. For this volume, I chose to make a cube whose side length was the distance of the most distant star in the sample.
- M$_\odot$ PC^{-3}: To find mass density, we may simply multiply our number density by the sum of the sample's masses. As instructed, the mass attribute for each source was calculated using the mass/luminosity relationship.
- GM CM^{-3}: This mass density can be calculated by scaling the above mass density by the relevant conversion factors, i.e. M$_\odot$-to-gm and (pc-to-cm)$^{-3}$.
- N_H CM^{-3}: To find the number density of hydrogen, I began by assuming that all stars were made up purely of hydrogen (i.e. $M_H = M_{\text{stars}}$). I then turned converted this value from solar masses to grams and then grams to number of hydrogen atoms. I then divided this number, as above, by the cube of the largest distance in the dataset. This yielded a value that was much higher (of order a couple thousand) than what I expected (around 1); not totally sure where this went wrong but I think it's likely just a conversion factor.

Problem 5. Prove that, if the orbital planes of binaries are oriented randomly with respect to the plane of the sky, then the average value of $\sin^3 i$ for the binaries is $\langle \sin^3 i \rangle = 0.59$.

Answer 5. We may approach this problem in one of two ways: we may numerically simulate observations or analytically solve it.

The analytical method seems to be the better path. To do so, we begin by recalling the definition of $\langle x \rangle$:

$$\langle x \rangle = \int x \; P(x) dx$$

Since we are looking for the average value of $\sin^3 i$, then this equation becomes

$$\langle \sin^3 i \rangle = \int_0^{\pi/2} \sin^3 i \; P(i) di$$

We set the upper bound of integration to $\pi/2$ to recognize that the appearance of disks with inclinations between $\pi/2$ and pi are degenerate with the appearance of disks in $[0, \pi/2]$. Letting $P(i) = \sin i$, we find

$$\begin{aligned} \langle \sin^3 i \rangle &= \int_0^{\pi/2} \sin^4 i di \\ &= \frac{3\pi}{16} = 0.59 \end{aligned}$$

Voila! We find, as expected, that $\langle \sin^3 i \rangle = 0.59$.

The numerical method would be to generate sample inclinations according to $P(i)$ and evaluate the resulting value of $\langle \sin^3 i \rangle$; taking the mean of that sample should return 0.59 as well.

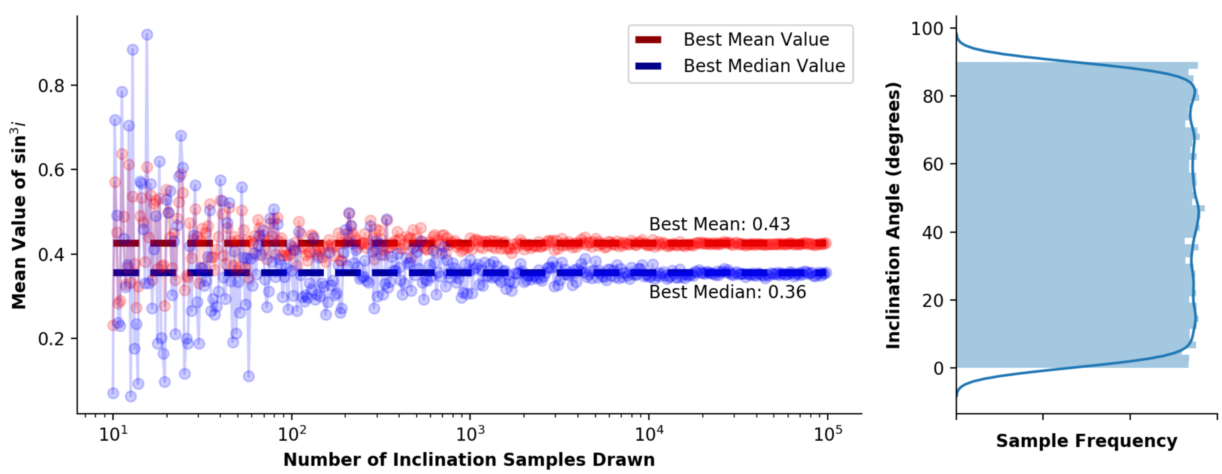


Figure 2: The distribution of mean values of $\langle \sin^3 i \rangle$ over various sample sizes.

Unfortunately, as shown in Fig. 2, this was not the case. The cause for the error in this plot is the characterization of the probability distribution. I chose a uniform distribution when writing this code (shown in the plot on the right - a flat, even sample of inclinations between 0 and 90^o), while it should be given by $\sin i$, which would push the mean higher and likely to 0.59, the expected value.

Problem 6. Show that, if the surface of a pulsar were at the same temperature as the photosphere of the Sun, the pulsar would have $\mathrm{M}_V \approx 30$.

Answer 6. We know that magnitude and luminosity are related by

$$M_1 - M_2 = -2.5 \log \frac{L_1}{L_2},$$

where $M_1, M_2, L_1, and L_2$ are the absolute magnitudes and luminosities of sources s_1 and s_2, respectively. By letting s_2 be the Sun, we may substitude in its known magnitude and luminosity, find L_1, and thus solve for M_1.

To do this, of course, we must solve for L_1. To do so, we may rearrange the effective temperature equation for luminosity, yielding $L = 4\pi R^2 \sigma T^4$. Since we are told that the temperature of our source is the same as that of the Sun, we may thus rewrite our initial equation as

$$\begin{aligned} M_{\text{pulsar}} &= M_\odot - 2.5 \log \frac{4\pi R_{\text{pulsar}}^2 \sigma T_\odot^4}{4\pi R_\odot^2 \sigma T_\odot^4} \\ &= M_\odot - 2.5 \log \left[\left(\frac{R_{\text{pulsar}}}{R_\odot} \right)^2 \right] \end{aligned}$$

Looking up values, we find that pulsars - which are neutron stars - have masses of just $\sim$10 km, making the ration of radii become $\frac{R_{\text{pulsar}}}{R_\odot} = 10/(7 \times 10^5) = 1.43 \times 10^{-5}$. Another lookup indicates that the Sun's absolute magnitude is $M_\odot = 4.83$. Therefore,

$$\begin{aligned} M_{\text{pulsar}} &= M_\odot - 2.5 \log \left[\left(\frac{R_{\text{pulsar}}}{R_\odot} \right)^2 \right] \\ &= 4.83 - 2.5 \log \left[\left(1.43 \times 10^{-5} \right)^2 \right] \\ &= 31.5 \end{aligned}$$

This shows that, unsurprisingly, very small objects would be very faint if they didn't have giant spinning laser beams periodically flashing at us.

```python
"""
Galactic Astronomy, HW3
Due: February 28
Jonas Powell
"""

import csv
import random
import numpy as np
import pandas as pd
import seaborn as sns
import astropy.units as u
import matplotlib.pyplot as plt
import astropy.constants as const
from sklearn.linear_model import LinearRegression
from astropy.coordinates import SkyCoord
from astroquery.vizier import Vizier
from astroquery.gaia import Gaia


def get_data(metric='distance', n_sources=1000):
    if metric not in ['distance', 'brightness']:
        return "Please choose 'distance' or 'brightness'."

    # Translate english to SQL/Gaia and call.
    param = 'parallax' if metric is 'distance' else 'lum_val'
    if metric is 'distance':
        # Want to remove sources closer than Alpha Cen, whose parallax angle is 0.768"
        gaia_str = "SELECT top {} * FROM gaiadr2.gaia_source \
                    WHERE parallax > 700 \
                    AND phot_rp_mean_mag != 'Nan' \
                    AND bp_rp != 'Nan' \
                    AND lum_val != 'Nan' \
                    ORDER BY parallax DESC \
                    ".format(n_sources)
    else:
        gaia_str = "SELECT top {} * FROM gaiadr2.gaia_source \
                    WHERE 'lum_val' > 0 \
                    AND phot_rp_mean_mag != 'Nan' \
                    AND bp_rp != 'Nan' \
                    AND lum_val != 'Nan' \
                    ORDER BY lum_val DESC \
                    ".format(n_sources)


    job = Gaia.launch_job(gaia_str)  #, dump_to_file=True)
    gaia_results_raw = job.get_results()

    gaia_results = gaia_results_raw.to_pandas()

    sort_feature = 'Distance' if metric is 'distance' else 'Absolute Magnitude'
```

```python
    df = pd.DataFrame()
    df['Distance'] = (gaia_results['parallax'] * 1e-3)**(-1)
    df['mag'] = gaia_results['phot_rp_mean_mag']
    df['Color'] = gaia_results['bp_rp']
    df['Absolute Magnitude'] = df['mag'] - \
        5 * (np.log10(df['Distance']) - 1)
    df['T Effective'] = gaia_results['teff_val']
    df['Parallax'] = gaia_results['parallax']
    df['Radius'] = gaia_results['radius_val']
    df['Luminosity'] = gaia_results['lum_val']
    df['Mass'] = df['Luminosity']**0.25
    df['Sort Feature'] = sort_feature
    df['Decimal Sort Feature'] = df[sort_feature]/np.nanmax(df[sort_feature])

    df.to_csv('stars_by_{}-{}.csv'.format(metric, n_sources))
    return df




# Problem 3
def make_plots(n_sources=1000, save=False):
    """Make the necessary plots for each dataset."""
    plt.clf()
    cmap1, cmap2 = 'Reds', 'Blues'

    # If we haven't already downloaded the data, get it.
    try:
        closest = pd.read_csv('stars_by_distance-{}.csv'.format(n_sources))
        print "Already have distance data; reading it in now."
    except IOError:
        closest = get_data(metric='distance', n_sources=n_sources)
        print "Don't yet have distance data yet; downloading now."

    try:
        brightest = pd.read_csv('stars_by_brightness-{}.csv'.format(n_sources)
    )
        print "Already have brightness data; reading it in now."
    except IOError:
        brightest = get_data(metric='brightness', n_sources=n_sources)
        print "Don't yet have brightness data yet; downloading now."

    df = pd.concat([closest, brightest])

    # Get the plots rolling.
    fig, ax = plt.subplots(figsize=(8, 8))

    # Shade by density
    ax = sns.kdeplot(closest['Color'], closest['Absolute Magnitude'],
                     cmap=cmap1, shade=True, shade_lowest=False)
    ax = sns.kdeplot(brightest['Color'], brightest['Absolute Magnitude'],
                     cmap=cmap2, shade=True, shade_lowest=False)

```

```
    # Plot contours
    # ax = sns.kdeplot(closest['Color'], closest['Absolute Magnitude'], n_levels=30, alpha=0.5, cmap=cmap1)
    # ax = sns.kdeplot(brightest['Color'], brightest['Absolute Magnitude'], n_levels=30, alpha=0.5, cmap=cmap2)

    # Scatter on the points
    ax = sns.scatterplot('Color', 'Absolute Magnitude', data=closest, ax=ax,
                         alpha=0.2, color='darkred', linewidth=0, label='Closest Stars')
    ax = sns.scatterplot('Color', 'Absolute Magnitude', data=brightest, ax=ax,
                         alpha=0.1, color='darkblue', linewidth=0, label='Brightest Stars')


    ax.set_xticks(np.linspace(min(df['Color']), max(df['Color']), 4))
    ax.set_yticks(np.linspace(min(df['Absolute Magnitude']),
                              max(df['Absolute Magnitude']), 4))

    ax.set_ylim(ax.get_xlim()[::-1])
    ax.set_title('HR Diagram', weight='bold')
    plt.legend(frameon=True)
    sns.despine()
    fig.tight_layout()
    fig.subplots_adjust(top=0.85, bottom=0.1)
    if save is True:
        outname = 'HR_diagram.pdf'
        plt.savefig(outname, dpi=200)
        print "Saved to", outname
    else:
        print "Showing:"
        plt.show(block=False)



# Problem 4
n_sources=1000
def get_densities(n_sources=1000):
    # If we haven't already downloaded the data, get it.
    try:
        df = pd.read_csv('stars_by_distance-{}.csv'.format(n_sources))
        print "Already have distance data; reading it in now."
    except IOError:
        df = get_data(metric='distance', n_sources=n_sources)
        print "Don't yet have distance data yet; downloading now."

    msol_to_gm = 1.989e33
    pc_to_cm = 3.08e18
    gm_to_nH = 1/(1.674e-24)
    n_stars = len(df['Distance'])

    n_per_pc3 = n_stars / np.nanmax(df['Distance'])**3
    m_sol_per_pc3 = np.sum(df['Mass']) / np.nanmax(df['Distance'])**3
```

```
    gm_per_cm3 = msol_to_gm * np.sum(df['Mass']) / (pc_to_cm * np.nanmax(df['Distance']))**3

    nH = np.sum(n_stars * df['Mass']) * msol_to_gm * gm_to_nH
    nH_per_cm3 = nH / (pc_to_cm * np.nanmax(df['Distance']))**3

    print "Stars per Cubic Parsec: ", n_per_pc3
    print "Solar Masses per Cubic Parsec: ", m_sol_per_pc3
    print "Grams per Cubic Centimeter: ", gm_per_cm3
    print "Hydrogen Atoms per Cubic Centimeter: ", nH_per_cm3
    print
    return [n_per_pc3, m_sol_per_pc3, gm_per_cm3, nH_per_cm3]


get_densities(n_sources=1000)




# Problem 5
def plot_inclinations(stepsize):
    plt.clf()

    def sample_inclinations(n_steps):
        inclinations, sin_cubed = [], []
        n = 0
        while n < n_steps:
            i = random.uniform(0, np.pi)
            sini = np.sin(i)
            inclinations.append(i)
            sin_cubed.append(sini**3)
            n += 1
        # return np.mean(sin_cubed)
        return (np.median(sin_cubed), np.mean(sin_cubed), inclinations)

    ns, means, medians, inclinations = [], [], [], []
    for n in np.arange(1, 5, stepsize):
        n_steps = 10**n
        ns.append(n_steps)
        median, mean, incls = sample_inclinations(n_steps)
        means.append(mean)
        medians.append(median)
        inclinations.append(incls)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4),
                                   gridspec_kw = {'width_ratios':[3, 1]})


    best_mean, best_median = means[-1], medians[-1]
    ax1.plot([ns[0], ns[-1]], [best_mean, best_mean], color='darkred',
             linewidth=4, linestyle='--', label='Best Mean Value')
    ax1.plot([ns[0], ns[-1]], [best_median, best_median], color='darkblue',
             linewidth=4, linestyle='--', label='Best Median Value')
```

```python

    ax1.semilogx(ns, means, '-or', alpha=0.2)
    ax1.semilogx(ns, medians, '-ob', alpha=0.2)

    ax1.text(10**4, 1.07 * best_mean, 'Best Mean: ' + str(round(best_mean, 2)))
    ax1.text(10**4, 0.83 * best_median, 'Best Median: ' + str(round(best_median, 2)))
    ax1.set_xlabel('Number of Inclination Samples Drawn', weight='bold')
    ax1.set_ylabel(r"Mean Value of sin$^3 i$", weight='bold')
    ax1.legend()

    i_deg = np.array(inclinations[-1]) * 90/np.pi
    sns.distplot(i_deg, vertical=True, ax=ax2)
    ax2.set_ylabel('Inclination Angle (degrees)', weight='bold')
    ax2.set_xlabel('Sample Frequency', weight='bold')
    start, end = ax2.get_xlim()
    ax2.xaxis.set_ticks(np.linspace(start, end, 4))
    ax2.set_xticklabels(ax2.get_xticklabels(), rotation=45)

    sns.despine()
    plt.tight_layout()

    plt.savefig('average_orbital_inclinations.png', dpi=200)
    plt.show()

    return i_deg




# The End
```