Continual Assessment Number 4

Introduction

Due to the nature of black holes, direct observations and measurements of black holes are almost impossible do but through the use of Kepler's law and a companion star, measurements such as the mass of a black hole can be measured through indirect observations using the companion star.
The aim of this report is to discover whether or not a black hole lies inside the close binary system GS2000 through the use of Kepler's law for binary systems, the measurement of the orbital period using the transit method and the measurement of the orbital velocity through the use of spectral analysis on the companion star.
As the primary object inside the binary system GS2000 is unknown, we shall henceforth call it a compact object.
In order to determine the first step in our calculation of the mass of the compact object, we should first observe the equation in which the mass of the compact object can be measured, which is the Kepler's law for binary system, which is derived through Kepler's third law and the equations of circular orbits.

Kepler's law for binary stars is given below :-

$$\frac{Mx^3 \sin^3 i}{(Mx+Mc)^2} = \frac{P K^3}{2\pi G} \quad (1)$$

Where Mx is the mass of the compact object, Mc is the mass of the companion star, i is the inclination of the orbital plane, P is the binary period and K is the orbital velocity of the companion star.

Through observing the emission line spectrum of the companion star, the radial velocity of the companion star can be found. Since the companion star is moving away and towards us when orbiting the compact object, the emission lines of the companion star will be shifted slightly when moving away and towards us. We can therefore compare the doppler shifted emission lines to emission lines of a template star, which can be described as a non moving star that is of a similar stellar type to the companion star. The radial velocity of the companion star is determined by trialling various radial velocities and doppler shifting the template star until the emission lines of both stars overlap one another.
The use of the emission line spectrum to determine the radial velocities instead of the overall spectrum of both companion and template stars is due to the overall shape of both spectrums are very much unknown and that the overlapping of both spectrums would become much more difficult.
By creating a plot of radial velocity with phase, the orbital velocity can be found by finding the semi-amplitude of the companion star through least squares fitting of the velocity curve.
The mass of the companion star is based on previous papers which state that the mass of the companion star is around 0.7 $M_\odot$ (Perrin, 1976). Along with the mass of the companion star, the binary period was also determined to be 0.3440915 days using the transit method (Chevalier & Ilovaisky, 1993).
Through the use of Monte-Carlo error propagation techniques, we can trial several inclines with the knowledge that the incline of the plane must be greater than 0 degrees, simply do to the fact that the mass of the compact object in equation 1 shoots to infinity as the incline falls to 0 degrees. With this same thought process we can estimate the minimum mass of the compact object by using 90 degrees as the incline of the plane.

Analysis

Cross-Correlations

Thirteen different spectra of GS2000 were produced by W. M. Keck Observatory, each different spectra relating to thirteen different binary phases.
Using the spectra of the companion star provided by W. M. Keck Observatory, we proceeded to cross-correlate the spectra of the companion star with a template star. The first hurdle in our cross-correlation was to decide on the stellar type of the template star since the stellar type of the companion star was unknown. Since the Keck Observatory also provided spectra of several different stellar types in which the absorption lines were clearly visible, we trialled and compared all possible stellar types with the companion star and used chi-squared minimisation to find the closest stellar type to the companion star.

Each template spectra followed similarly to Figure 1 but with a slightly different overall shape and absorption lines.

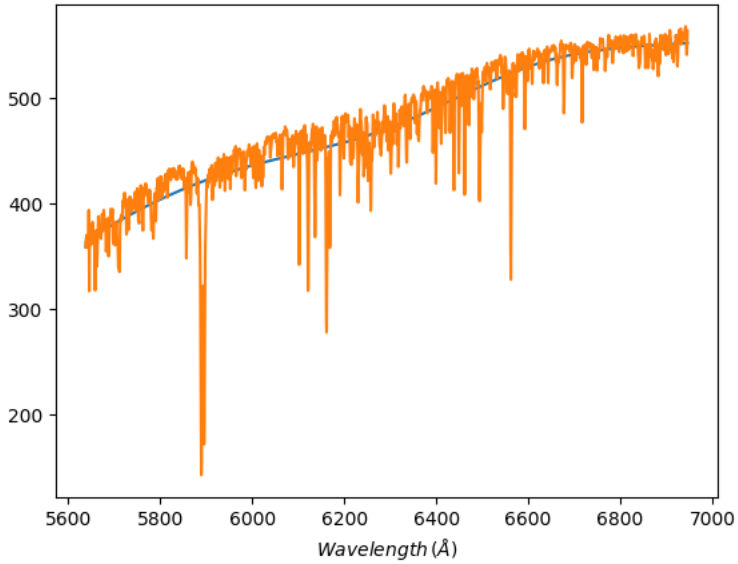

Figure 1 : Spectra of a stellar type k5 star provided by W. M. Keck Observatory along with a fit of the overall shape of the spectra using a cubic spline.

From Figure 1, it is clear that the emission line spectrum of the template star can be obtained through removing the overall shape of the spectrum from the spectra so that the absorption lines in Figure 1 become emission lines as seen in Figure 2. In order to obtain the overall shape of the spectrum a cubic spline was used with a small number of knots to create a smooth fitting function as can be seen in Figure 1.
Since the cubic spline function used in our report was from python's scipy package includes an interpolation function we can find the flux values of the fit corresponding to the wavelengths of the template spectrum. From this we can simply remove the flux of the fitted function from the flux of the spectra, and this results in the emission line spectrum found in Figure 2.

The same thing can be done for the GS2000 companion star, which can be seen in Figures 3 and 4.

Since the GS2000 companion star's emission line spectrum is doppler shifted, we doppler shifted the template star with different trial velocity with the 'brute-force' method until both spectrums have overlapping emission lines. As the emission line spectrum for the template star lacks the dip in the flux at around 6600 angstroms that the companion star possesses, we chose to limit the spectrum from 5700 to 6200 angstroms. In order to choose the best possible velocity, we used chi-squared minimisation and repeated this process with all the possible template stars.

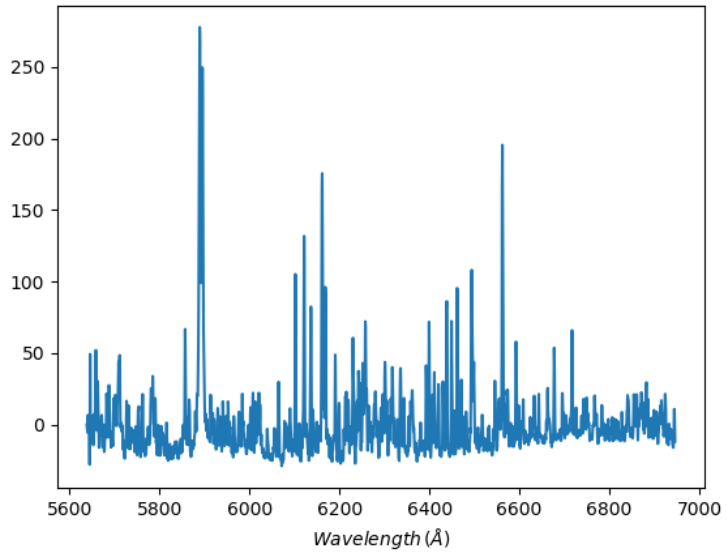

Figure 2 : Emission line spectrum of a k5 stellar type star.

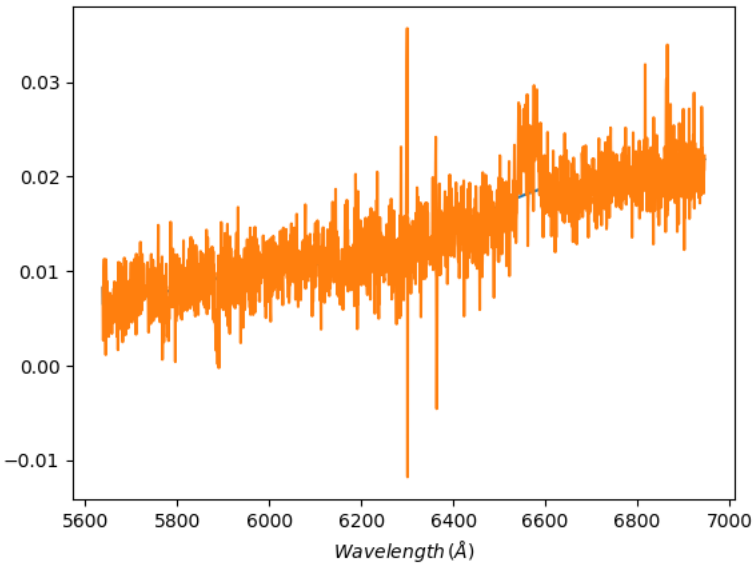

Figure 3 : Spectra of a GS2000 companion star at a phase of -0.1405
provided by W. M. Keck Observatory along with a fit of the overall shape of the spectra using a cubic spline.

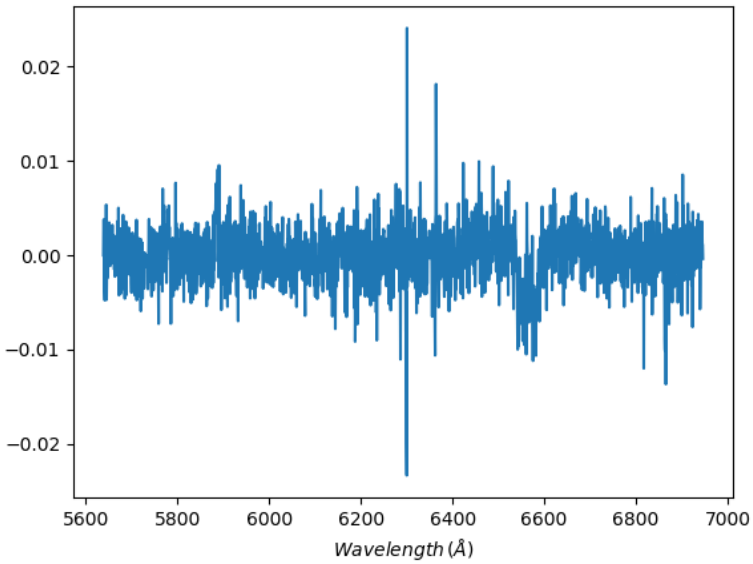

Figure 4 : Emission line spectrum of GS2000 companion star at a phase of -0.1405.

The equation used to doppler shift the wavelength was through the following :-

$$\lambda_{new} = \lambda_{orig}\left(1 + \frac{v}{c}\right) \tag{2}$$

Where λ_{new} is the doppler shifted wavelength, λ_{orig} is the original un-moved wavelength, v is the trial velocity and c is the speed of light.

To use the chi-squared minimisation we needed to first scale the data, this was done using chi-squared itself and a scaling parameter :-

$$\chi^2 = \sum_i^N \left(\frac{y_i - AP(x_i)}{\sigma_i}\right)^2 \tag{3}$$

Where A is the scaling parameter, y_i is the data we want to scale to, $P(x_i)$ is the pattern we want to scale and σ_i is the error in the flux of the companion star which was also provided by W. M. Keck Observatory.
Therefore our y_i is the GS2000 companion star data and $P(x_i)$ is the template star data. To get the best possible scaling parameter, we must differentiate chi-squared with respect to the scaling parameter, which leads to :-

$$A = \frac{\sum_i^N y_i P(x_i)/\sigma_i^2}{\sum_i^N [P(x_i)]^2/\sigma_i^2} \tag{4}$$

Using this scaling parameter we can now find the chi-squared values at different trial velocities, repeating this for all possible template stars leads to Figure 5.

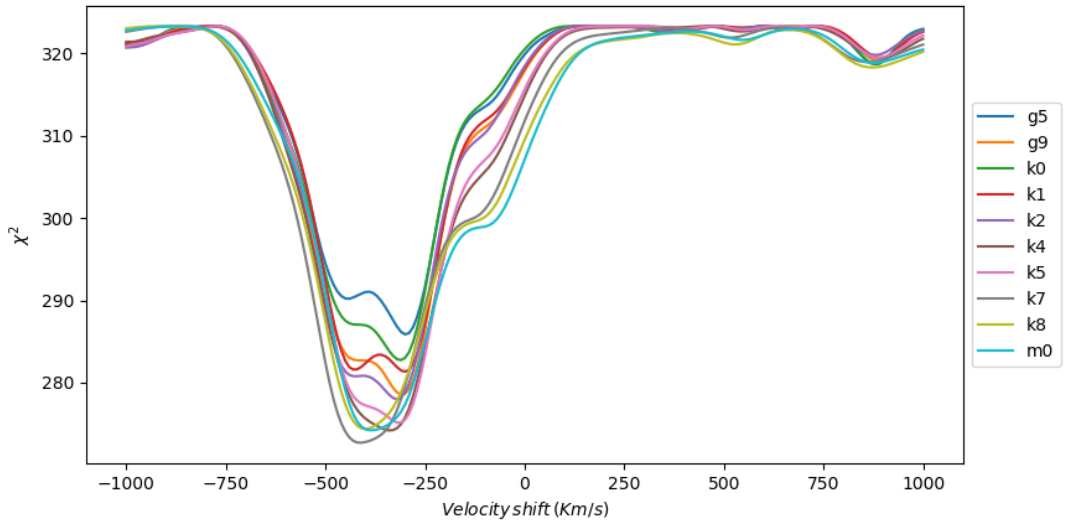

Figure 5 : A chi-squared plot with the companion star at a phase of -0.1405 and with all the possible template stars labelled above.

From Figure 5 we can see that all provided stellar type star have a similar chi-squared which could be due to the considerable noise present in the emission line spectrum of the companion star, but no considerable improvement was observed to the chi-squared plot when the application of a central moving average with a window width of significant length was applied to the spectra of the companion star.
Of all the stellar types, k5 produced the most consistently lowest chi-squared values when repeated for all 13 phases of the companion star.
Using Figure 6 we can see that in all 13 phases chi squared reaches a minima at a certain velocity. From equation 3 we can see that at this minima, the emission line spectrum of the companion is overlapping the scaled doppler shifted emission line spectrum of the template star and therefore at these minimas, the best possible value for the radial velocity can be found.

A plot of radial velocity with the phases of the companion star can be made as seen in Figure 7, and using the following equation a velocity curve can be fitted :-

$$V(\phi) = \gamma + K_X \sin(2\pi\phi) + K_Y \cos(2\pi\phi) \quad (5)$$

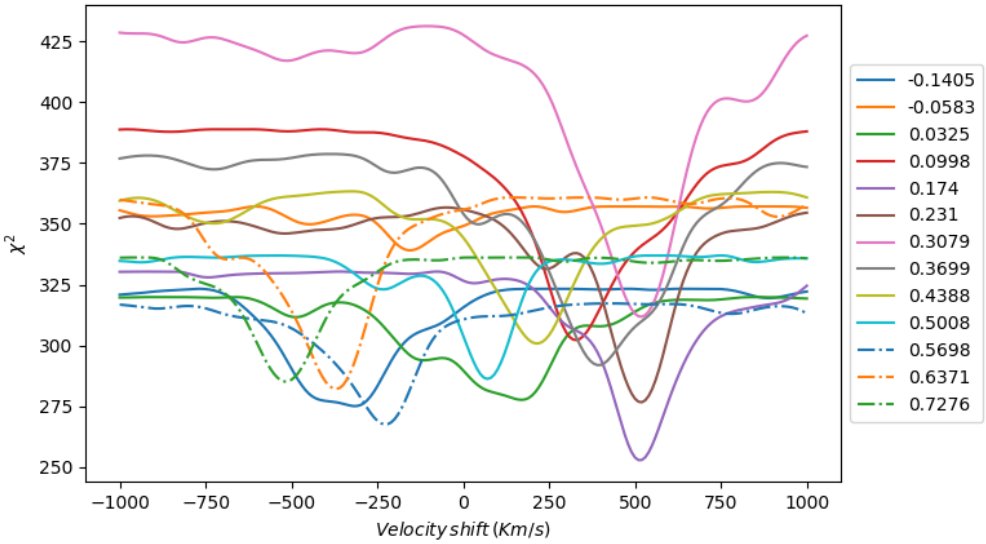

Figure 6 : A chi-squared plot with the companion star at a all 13 phases which are labelled above and with k5 star as the template star.

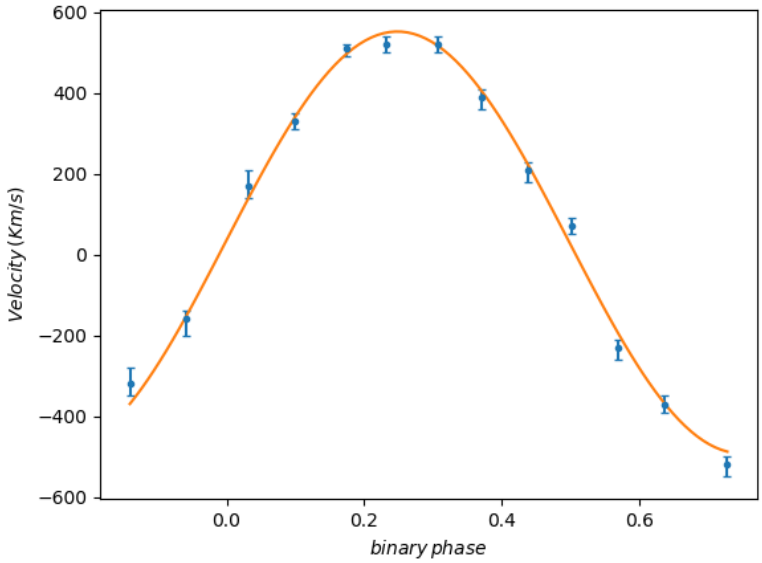

Figure 7 : A plot of radial velocity with the phases of the companion star and a least squares fit of a velocity curve.

To fit the velocity curve to the radial velocity and phase data we must find the optimal values of γ, K_X and K_Y. To do this we must use chi-squared minimisation and therefore differentiate chi-squared with respect to all three parameters, this in turn leads to three simultaneous equations which are incredibly difficult to solve analytically. But through turning the simultaneous equations into a Hessian matrix, we can solve for all three parameters and the equation used to solve for the parameters can be seen in the appendix.

By multiplying the inverse Hessian matrix with the right hand side of the equation all three parameters were determined as follows; γ was determined to be 29.94+/-6.84 kms^{-1}, K_X was determined to be 522.04+/-9.41 kms^{-1} and K_Y was determined to be 5.89+/-9.57 kms^{-1}.

Using these parameters, the semi-amplitude of the radial velocity can be found and hence the orbital velocity of the companion star can be determined, the equation for the semi-amplitude of the radial velocity is given below :-

$$K = \left(K_X{}^2 + K_Y{}^2 \right)^{1/2} \quad (6)$$

Where K is the semi-amplitude of the radial velocity and also the orbital velocity.

From equation 6, the orbital velocity of the companion star was determined to be 522.08+/-9.38 kms^{-1}.
To make the determination of the mass of the compact object easier, we decide to bring an intermediate function for equation 1 which is called the mass function, $f(M_X)$:-

$$f\left(M_X\right) = \frac{Mx^3 \sin^3 i}{\left(Mx + Mc\right)^2} = \frac{P K^3}{2\pi G} \quad (7)$$

Using the orbital velocity of 522.08 kms^{-1} and binary period of 0.3440915 days, the mass function, $f(M_X)$ was determined to be 5.08+/-0.27 $M_\odot$.

We can find the mass of the compact object for various different angles for the incline of the plane using the mass function and equation 7. Using *i* as 90 degrees, we can find the minimum mass of the compact star, $M_{X\,min}$.
The typical mass of a k5 stellar type star is 0.7 $M_\odot$ and therefore using this value and *i* as 90 degrees, the minimum mass of the compact object, $M_{X\,min}$ was determined to be 5.17+/-0.26 $M_\odot$.

Since several parameters such as the incline and the companion star mass are not entirely known, we must use Monte-Carlo error propagation to consider all possibilities of these unknown parameters.

We therefore assume that the angle must be above somewhere around 70 degrees, as below this value the mass of the compact object becomes much greater than the Schwarzschild limit of 3 $M_\odot$ and is certain of being a black hole.
We assume an uncertainty of around 40% for the mass of the companion star since the mass of a k type star is typically between 0.4 and 0.8 $M_\odot$.
An assumption of an uncertainty of 10% for the binary period since the binary period was previously known through transit method, which can be a fairly accurate method of determining the binary period.

Using the Monte-Carlo error propagation the mass of the compact object, M_X was found to be 5.67+/-1.76 $M_\odot$. This value of 5.67+/-1.76 $M_\odot$ is comparable and within error to the value of 5.9 – 7.5 $M_\odot$ which was determined by Filippenko, Matheson, & Barth (1995). Using the cumulative density function found in Figure 8 the probability that the mass of the compact object is greater than 3 $M_\odot$ was determined to be 96.2%, therefore we can say with 96% confidence that the compact object is indeed a black hole.

Conclusion

Through spectral analysis the orbital velocity of the companion star was determined to be 522.08+/-9.38 kms^{-1}. The minimum mass of the compact object, $M_{X\,min}$ was determined to be 5.17+/-0.26 $M_\odot$ using a mass of 0.7 $M_\odot$ for the companion star and an incline of 90 degrees.
Using the Monte-Carlo error propagation the mass of the compact object, M_X was found to be 5.67+/-1.76 $M_\odot$. With 96% confidence we can say that the compact object is indeed a black hole.

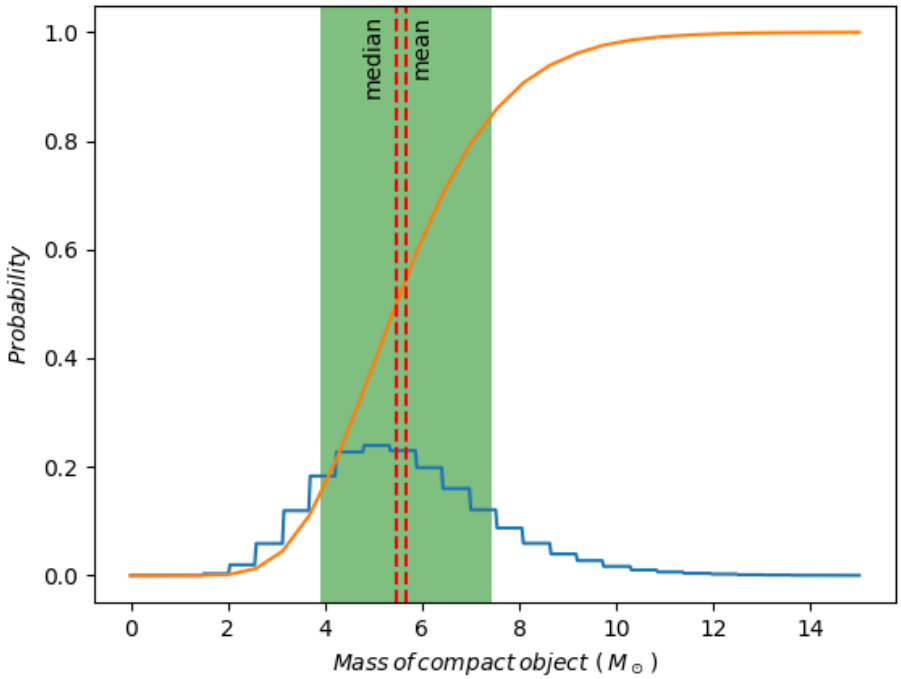

Figure 8 : A plot of probability density and cumulative density functions for the mass of the compact object, where the orange line represents the cumulative density function, the blue line represents the probability density function, the two red dashed lines represent both the mean and median of M_X and are labelled appropriately and finally the shaded green region represents 1-sigma confidence interval for M_X.

References

Chevalier, C., & Ilovaisky, S. (1993). Optical studies of transient low-mass X-ray binaries. IV - A 10-hour distortion wave in the quiescent light curve of GS 2000 + 25, *269*, 301–309.

Filippenko, A., Matheson, T., & Barth, A. (1995). A Black Hole in the X-Ray Nova GS 2000+25, *455*, L139.

Perrin, M. (1976). Mass Estimation of Twelve K Dwarfs. In B. Hauck, P. Keenan, & W. Morgan (Eds.), *Abundance Effects in Classification* (Vol. 72, p. 167).

Appendix

Hessian Matrix

$$\begin{bmatrix} \sum_i^N w & \sum_i^N w\sin(2\pi\phi_i) & \sum_i^N w\cos(2\pi\phi_i) \\ \sum_i^N w\sin(2\pi\phi_i) & \sum_i^N w\sin^2(2\pi\phi_i) & \sum_i^N w\sin(2\pi\phi_i)\cos(2\pi\phi_i) \\ \sum_i^N w\cos(2\pi\phi_i) & \sum_i^N w\sin(2\pi\phi_i)\cos(2\pi\phi_i) & \sum_i^N w\cos^2(2\pi\phi_i) \end{bmatrix} \begin{bmatrix} \gamma \\ K_X \\ K_Y \end{bmatrix} = \begin{bmatrix} \sum_i^N w v_i \\ \sum_i^N w\sin(2\pi\phi_i)v_i \\ \sum_i^N w\cos(2\pi\phi_i)v_i \end{bmatrix}$$

Where w is the weight and is equal to $1/\sigma^2$, φ is the phase and ν is the radial velocity.

Code

-*- coding: utf-8 -*-

```
"""
Created on Wed Oct 04 10:57:34 2017
@author: Gerwyn
"""
from __future__ import division
import numpy as np
import scipy as sp
import scipy.interpolate as spi
import scipy.stats as sps
import matplotlib.pyplot as plt
import uncertainties as u
from uncertainties import unumpy
plt.close('all')
@u.wrap
def roots(n=0, *P):
    """ Adapted the numpy.roots function to take into account the
uncertainties in the Coefficients """
    p = np.array(P)
    N = len(p)
    M = np.diag(np.ones((N - 2,), p.dtype), -1)
    M[0, :] = -p[1:] / p[0]
    r = np.linalg.eigvals(M)
    return r[n]
def cubic_spline(x, x_data, y_data, s_data, t_data, per_data):
    """ Cubic Spline """
    tck = spi.splrep(x_data, y_data, k=3, s=s_data, t=t_data,
per=per_data)
    return spi.splev(x, tck)
def smooth(y_data, window_width):
    """ Smoothing data """
    y_smooth = np.zeros(len(y_data))
    for i in range(len(y_data)):
        if i > len(y_data) - window_width / 2:
            S = y_data[i - int(window_width / 2):]
            y_smooth[i] = np.sum(S)/len(S)
        elif i < window_width / 2:
            S = y_data[: i + int(window_width / 2) + 1]
            y_smooth[i] = np.sum(S)/len(S)
        else:
            S = y_data[i - int(window_width / 2): i + int(window_width /
2) + 1]
            y_smooth[i] = np.sum(S)/len(S)
    return y_smooth
def doppler_shift(lambda_original, velocity):
    """ Velocity in terms of ms-1 """
    c = 3e8
    lambda_new = lambda_original*(1 + velocity/c)
    return lambda_new
def sinusoidal_curve(gamma, Kx, Ky, phase):
    return gamma + Kx * np.sin(2 * np.pi * phase) + Ky * np.cos(2 *
np.pi * phase)
```

```
def f_Mx_func(P, K):
    P_d = P*(24*60*60)
    G = 6.67e-11
    return (P_d*(K*1e3)**3)/(2*np.pi*G)
def Mx_func(Mc, i, f_Mx, Decision=0):
    f_Mx_val = unumpy.nominal_values(f_Mx)
    c1 = (np.sin(i))**3
    c2 = -f_Mx_val
    c3 = -2*f_Mx_val
    c4 = -Mc**2*f_Mx_val
    Coeff_real = [c1, c2, c3, c4]
    Root = np.roots(Coeff_real)
    Root_very_real = Root[Root.imag == 0].real
    if Decision == 0:
        Result = Root_very_real
    else:
        f_Mx_std = unumpy.std_devs(f_Mx)
        sc1 = 0
        sc2 = f_Mx_std
        sc3 = 2*f_Mx_std
        sc4 = f_Mx_std
        C1 = u.ufloat(c1, sc1)
        C2 = u.ufloat(c2, sc2)
        C3 = u.ufloat(c3, sc3)
        C4 = u.ufloat(c4, sc4)
        Coeff = [C1, C2, C3, C4]
        for m in xrange(len(Coeff) - 1):
            Root = roots(m, *Coeff)
            if Root.nominal_value == Root_very_real:
                Result = Root
            else:
                continue
    return Result
# setting up the template star
template_list = ['k5']
gs200_id = np.arange(1, 14)
gs200_size = gs200_id.size
# Creating possible trial velocities
v = 1e6
Vel = np.linspace(-v, v, 201)
# Creating arrays for the radial velocities and their errors
radial_vel = np.zeros(gs200_size)
error_rad_vel_less = np.zeros(gs200_size)
error_rad_vel_more = np.zeros(gs200_size)
# Creating an array for the companion star phase
phase = np.array([-0.1405, -0.0583, 0.0325, 0.0998, 0.1740, 0.2310,
0.3079, 0.3699, 0.4388, 0.5008, 0.5698, 0.6371, 0.7276])
# Choosing cut-off wavelengths
Wavelength_min = 5700
Wavelength_max = 6200
linestyle = ['solid', 'solid', 'solid', 'solid', 'solid', 'solid',
'solid', 'solid', 'solid', 'solid', 'dashdot', 'dashdot', 'dashdot']
# Reading template file
W = str(template_list[0])
```

```
lambda_temp_orig, flux_temp_orig, err_temp_orig = np.loadtxt(
    '/home/gerwyn/Documents/Physics/Computing Year 4/Data-
Analysis/Assignments/Assignment
4/MiniProjectAllData/KeckTemplates/keck_' + W + '.txt',
    unpack=True)
# Define h:
window_width = 0    # 0 = Not Smoothed
# Create an array of smoothed data:
flux_temp_smooth = smooth(flux_temp_orig, window_width)
# Define the number of knots:
nknots = 4
# Create the array of knots:
knots = np.arange(lambda_temp_orig[1], lambda_temp_orig[-1],
                  (lambda_temp_orig[-1] - lambda_temp_orig[1]) /
np.double(nknots))
# Use cubic spline:
spl_temp = cubic_spline(lambda_temp_orig, lambda_temp_orig,
flux_temp_smooth, s_data=None, t_data=knots, per_data=0)
# Continuum subtracted template:
continuum_subtracted_temp = spl_temp - flux_temp_smooth
# Plotting spectra
plt.figure()
plt.plot(lambda_temp_orig, spl_temp)
plt.plot(lambda_temp_orig, flux_temp_orig)
plt.xlabel(r"$Wavelength \/ (\AA)$")
plt.figure()
plt.plot(lambda_temp_orig, continuum_subtracted_temp)
plt.xlabel(r"$Wavelength \/ (\AA)$")
fig1, ax1 = plt.subplots()
for j in range(gs200_size):
    id = gs200_id[j]
    if id < 10:
        # Reading GS2000 files
        Q = str(id)
        lambda_keck_gs2000_orig, flux_keck_gs2000_orig,
err_keck_gs2000_orig = np.loadtxt(
            '/home/gerwyn/Documents/Physics/Computing Year 4/Data-
Analysis/Assignments/Assignment
4/MiniProjectAllData//GS2000/keck_gs2000_0' + Q + '.txt',
            unpack=True)
        # Create an array of smoothed data:
        flux_keck_gs2000_smooth = smooth(flux_keck_gs2000_orig,
window_width)
        # Create the array of knots:
        knots = np.arange(lambda_keck_gs2000_orig[1],
lambda_keck_gs2000_orig[-1],
                          (lambda_keck_gs2000_orig[-1] -
lambda_keck_gs2000_orig[1]) / np.double(nknots))
        # Use cubic spline:
        spl_keck = cubic_spline(lambda_keck_gs2000_orig,
lambda_keck_gs2000_orig, flux_keck_gs2000_smooth, s_data=None,
                    t_data=knots,
                    per_data=0)
        # Continuum subtracted gs200:
```

```
        continuum_subtracted_keck = spl_keck - flux_keck_gs2000_smooth
        # Creating arrays for the parameters and their errors
        A_parameter = np.zeros(len(Vel))
        sigma_A = np.zeros(len(Vel))
        Chi_squared = np.zeros(len(Vel))
        for i in xrange(len(Vel)):
            # Doppler shift
            doppler_lambda_temp = doppler_shift(lambda_temp_orig,
Vel[i])
            # Interpolating template using cubic spline
            continuum_temp_shifted =
cubic_spline(lambda_keck_gs2000_orig, doppler_lambda_temp,
continuum_subtracted_temp, s_data=None, t_data=None, per_data=0)
            # Reducing the arrays so the may be interpolated correctly
through cubic splines and exclude difference in patterns above 6400
            # Keck arrays
            reduced_lambda_keck = lambda_keck_gs2000_orig[
                (lambda_keck_gs2000_orig >= Wavelength_min) &
(lambda_keck_gs2000_orig <= Wavelength_max)]
            reduced_continuum_keck = continuum_subtracted_keck[
                (lambda_keck_gs2000_orig >= Wavelength_min) &
(lambda_keck_gs2000_orig <= Wavelength_max)]
            reduced_err_keck_gs2000 = err_keck_gs2000_orig[
                (lambda_keck_gs2000_orig >= Wavelength_min) &
(lambda_keck_gs2000_orig <= Wavelength_max)]
            # Template arrays
            reduced_continuum_temp_shifted = continuum_temp_shifted[
                (lambda_keck_gs2000_orig >= Wavelength_min) &
(lambda_keck_gs2000_orig <= Wavelength_max)]
            # Scaling parameter
            A_parameter[i] = np.sum((reduced_continuum_keck *
reduced_continuum_temp_shifted) / (
                    reduced_err_keck_gs2000 ** 2)) / np.sum(
                    (reduced_continuum_temp_shifted ** 2) /
(reduced_err_keck_gs2000 ** 2))
            sigma_A[i] = 1 / np.sum((reduced_continuum_temp_shifted **
2) / (reduced_err_keck_gs2000 ** 2))
            # Chi Squared
            Chi_squared[i] = np.sum((reduced_continuum_keck -
reduced_continuum_temp_shifted * A_parameter[i]) ** 2 /
                                    reduced_err_keck_gs2000 ** 2)
        if id == 1:
            index = np.argmin(Chi_squared)
            A = u.ufloat(A_parameter[index], np.abs(sigma_A[index]))
            print 'A parameter : {:.2u}'.format(A)
            # Plotting spectra
            fig2, ax2 = plt.subplots()
            ax2.plot(lambda_keck_gs2000_orig, spl_keck)
            ax2.plot(lambda_keck_gs2000_orig, flux_keck_gs2000_smooth)
            ax2.set_xlabel(r"$Wavelength \/ (\AA)$")
            fig4, ax4 = plt.subplots()
            ax4.plot(lambda_keck_gs2000_orig, continuum_subtracted_keck)
            ax4.set_xlabel(r"$Wavelength \/ (\AA)$")
        else:
```

```
            Carry_on = 0
        # Plotting chi-squared
        ax1.plot(Vel/(1e3), Chi_squared, label=phase[j],
linestyle=linestyle[j])
        ax1.set_xlabel(r"$Velocity \/ shift \/ (Km/s)$")
        ax1.set_ylabel(r"$\chi^2$")
        index = np.argmin(Chi_squared)
        Chi_min = Chi_squared[index]
        radial_vel[j] = Vel[Chi_squared == Chi_min]
        Chi_plus_one = Chi_min + 1
        less_than_chi = Chi_squared[Vel < radial_vel[j]]
        more_than_chi = Chi_squared[Vel > radial_vel[j]]
        less_than_vel = Vel[Vel < radial_vel[j]]
        more_than_vel = Vel[Vel > radial_vel[j]]
        error_rad_vel_less[j] = np.max(less_than_vel[less_than_chi >
Chi_plus_one]) - radial_vel[j]
        error_rad_vel_more[j] = radial_vel[j] -
np.min(more_than_vel[more_than_chi > Chi_plus_one])
    elif id >= 10:
        Q = str(id)
        lambda_keck_gs2000_orig, flux_keck_gs2000_orig,
err_keck_gs2000_orig = np.loadtxt(
            '/home/gerwyn/Documents/Physics/Computing Year 4/Data-
Analysis/Assignments/Assignment
4/MiniProjectAllData//GS2000/keck_gs2000_' + Q + '.txt',
            unpack=True)
        flux_keck_gs2000_smooth = smooth(flux_keck_gs2000_orig,
window_width)
        # Create the array of knots:
        knots = np.arange(lambda_keck_gs2000_orig[1],
lambda_keck_gs2000_orig[-1],
                          (lambda_keck_gs2000_orig[-1] -
lambda_keck_gs2000_orig[1]) / np.double(nknots))
        # Use cubic spline:
        spl_keck = cubic_spline(lambda_keck_gs2000_orig,
lambda_keck_gs2000_orig, flux_keck_gs2000_smooth, s_data=None,
                    t_data=knots,
                    per_data=0)
        # Continuum subtracted gs200:
        continuum_subtracted_keck = spl_keck - flux_keck_gs2000_smooth
        A_parameter = np.zeros(len(Vel))
        sigma_A = np.zeros(len(Vel))
        Chi_squared = np.zeros(len(Vel))
        for i in xrange(len(Vel)):
            # Doppler shift
            doppler_lambda_temp = doppler_shift(lambda_temp_orig,
Vel[i])
            # Interpolating template using cubic spline
            continuum_temp_shifted =
cubic_spline(lambda_keck_gs2000_orig, doppler_lambda_temp,
continuum_subtracted_temp, s_data=None, t_data=None, per_data=0)
            # Reducing the arrays so the may be interpolated correctly
through cubic splines and exclude difference in patterns above 6400
            # Keck arrays
```

```
            reduced_lambda_keck = lambda_keck_gs2000_orig[
                (lambda_keck_gs2000_orig >= Wavelength_min) &
(lambda_keck_gs2000_orig <= Wavelength_max)]
            reduced_continuum_keck = continuum_subtracted_keck[
                (lambda_keck_gs2000_orig >= Wavelength_min) &
(lambda_keck_gs2000_orig <= Wavelength_max)]
            reduced_err_keck_gs2000 = err_keck_gs2000_orig[
                (lambda_keck_gs2000_orig >= Wavelength_min) &
(lambda_keck_gs2000_orig <= Wavelength_max)]
            # Template arrays
            reduced_continuum_temp_shifted = continuum_temp_shifted[
                (lambda_keck_gs2000_orig >= Wavelength_min) &
(lambda_keck_gs2000_orig <= Wavelength_max)]
            # Scaling parameter
            A_parameter[i] = np.sum((reduced_continuum_keck *
reduced_continuum_temp_shifted) / (
                    reduced_err_keck_gs2000 ** 2)) / np.sum(
                    (reduced_continuum_temp_shifted ** 2) /
(reduced_err_keck_gs2000 ** 2))
            sigma_A[i] = 1 / np.sum((reduced_continuum_temp_shifted **
2) / (reduced_err_keck_gs2000 ** 2))
            # Chi Squared
            Chi_squared[i] = np.sum((reduced_continuum_keck -
reduced_continuum_temp_shifted * A_parameter[i]) ** 2 /
                                    reduced_err_keck_gs2000 ** 2)
        # Plotting chi-squared
        ax1.plot(Vel/(1e3), Chi_squared, label=phase[j],
linestyle=linestyle[j])
        ax1.set_xlabel(r"$Velocity \/ shift \/ (Km/s)$")
        ax1.set_ylabel(r"$\chi^2$")
        index = np.argmin(Chi_squared)
        Chi_min = Chi_squared[index]
        radial_vel[j] = Vel[Chi_squared == Chi_min]
        Chi_plus_one = Chi_min + 1
        less_than_chi = Chi_squared[Vel < radial_vel[j]]
        more_than_chi = Chi_squared[Vel > radial_vel[j]]
        less_than_vel = Vel[Vel < radial_vel[j]]
        more_than_vel = Vel[Vel > radial_vel[j]]
        error_rad_vel_less[j] = np.max(less_than_vel[less_than_chi >
Chi_plus_one]) - radial_vel[j]
        error_rad_vel_more[j] = radial_vel[j] -
np.min(more_than_vel[more_than_chi > Chi_plus_one])
# Shrink current axis by 20%
box = ax1.get_position()
ax1.set_position([box.x0, box.y0, box.width * 0.8, box.height])
# Put a legend to the right of the current axis
ax1.legend(loc='center left', bbox_to_anchor=(1, 0.5))
Average_sig = (error_rad_vel_more + error_rad_vel_less) / 2
# Creating our Hessian Matrix and solving to find our best fit
parameters
w = 1 / Average_sig ** 2   # weight
W = np.sum(w)
Wsin = np.sum(w * np.sin(2 * np.pi * phase))
Wcos = np.sum(w * np.cos(2 * np.pi * phase))
```

```
Wsin2 = np.sum(w * (np.sin(2 * np.pi * phase)) ** 2)
Wcos2 = np.sum(w * (np.cos(2 * np.pi * phase)) ** 2)
Wcossin = np.sum(w * np.cos(2 * np.pi * phase) * np.sin(2 * np.pi *
phase))
Hessian_Matrix = np.array([W, Wsin, Wcos, Wsin, Wsin2, Wcossin, Wcos,
Wcossin, Wcos2]).reshape(3, 3)
vel = unumpy.uarray(radial_vel[:], np.abs(Average_sig))
Wv = np.sum(w * vel)
Wvsin = np.sum(w * np.sin(2 * np.pi * phase) * vel)
Wvcos = np.sum(w * np.cos(2 * np.pi * phase) * vel)
Vel_Matrix = np.array([Wv, Wvsin, Wvcos]).reshape(3, 1)
Inverse_Hessian = np.linalg.inv(Hessian_Matrix)
Dot_Product = np.dot(Inverse_Hessian, Vel_Matrix)
# Determining our curve fit parameters
gamma = Dot_Product[0] / (1e3)
Kx = Dot_Product[1] / (1e3)
Ky = Dot_Product[2] / (1e3)
print "gamma : ", gamma
print "Kx : ", Kx
print "Ky : ", Ky
# Plotting radial velocity and phase
phase_trial = np.linspace(np.min(phase), np.max(phase), 100)
sinusoidal_vel = sinusoidal_curve(gamma, Kx, Ky, phase_trial)
K = (Kx ** 2 + Ky ** 2) ** (1 / 2)
ls = 'None'
fig, ax = plt.subplots()
# standard error bars
ax.errorbar(phase, radial_vel[:] / (1e3),
            yerr=[error_rad_vel_less[:] / (1e3), error_rad_vel_more[:] /
(1e3)],
            marker=".", linestyle=ls, capsize=2)
ax.set_ylabel(r"$Velocity \/ (Km/s)$")
ax.set_xlabel(r"$binary \/ phase$")
Curve_val = unumpy.nominal_values(sinusoidal_vel)
ax.plot(phase_trial, Curve_val)
K = (Kx ** 2 + Ky ** 2) ** (1 / 2)
K_vector = u.ufloat(unumpy.nominal_values(K), unumpy.std_devs(K))
# Defining our parameters
P = 0.3440915  # units of days
M_sun = 1.989e30
# Mass function
f_Mx = f_Mx_func(P, K)
# Finding minimum mass using i = 90 degrees and Mc = 0.7
incline = np.deg2rad(90)
Mc = 0.7*M_sun
Mx_i = Mx_func(Mc, incline, f_Mx, 1)
Mx_i_sol_unit = Mx_i/M_sun
print 'K : {:.2f}'.format(K_vector)
f_Mx_sol_unit = f_Mx/M_sun
f_Mx_sol_unit_vector = u.ufloat(unumpy.nominal_values(f_Mx_sol_unit),
unumpy.std_devs(f_Mx_sol_unit))
print 'f(Mx) : {:.2f}'.format(f_Mx_sol_unit_vector)
print 'Mx : {:.2f}'.format(Mx_i_sol_unit)
""" Monte-carlo eror propagation """
# Number of monte carlo trials
```

```
Ntrial = 1e5
# Creating standard deviations for each parameter
standard_Mc = 0.4*Mc
standard_K = 0.1*K_vector.nominal_value
standard_P = 0.1*P
angle = np.linspace(np.deg2rad(70), np.deg2rad(90), 1e4)
# Creating arrays for paramaters and Mx
incline_trial = np.zeros(np.int(Ntrial))
Mc_trial = np.zeros(np.int(Ntrial))
K_trial = np.zeros(np.int(Ntrial))
P_trial = np.zeros(np.int(Ntrial))
f_Mx_trial = np.zeros(np.int(Ntrial))
Mx_trial = np.zeros(np.int(Ntrial))
for l in xrange(np.int(Ntrial)):
    incline_trial[l] = np.random.choice(angle)
    Mc_trial[l] = np.random.normal(Mc, standard_Mc)
    K_trial[l] = np.random.normal(unumpy.nominal_values(K), standard_K)
    P_trial[l] = np.random.normal(P, standard_P)
    f_Mx_trial[l] = f_Mx_func(P_trial[l], K_trial[l])
    Mx_trial[l] = Mx_func(Mc_trial[l], incline_trial[l], f_Mx_trial[l],
0)
Mx_sol_unit = Mx_trial / M_sun
Mx_mean = np.mean(Mx_sol_unit)
Mx_median = np.median(Mx_sol_unit)
Mx_std = np.std(Mx_sol_unit)
Mx = u.ufloat(Mx_mean, Mx_std)
print 'Monte-Carlo Mx : {:.2f}'.format(Mx)
# Plotting probabilities
hist = np.histogram(Mx_sol_unit, bins=30)
hist_dist = sps.rv_histogram(hist)
X = np.linspace(0.0, 15, 500)
plt.figure()
plt.plot(X, hist_dist.pdf(X), label='PDF')
plt.plot(X, hist_dist.cdf(X), label='CDF')
xcoords = [Mx_mean, Mx_median]
text_coords = [Mx_mean + 0.1, Mx_median - 0.7]
plot_text = ['mean', 'median']
colour = ['red', 'red']
plot_index = 0
for xc in xcoords:
    plt.axvline(x=xc, color=colour[plot_index], linestyle='--')
    plt.text(text_coords[plot_index], 1, plot_text[plot_index],
rotation=90)
    plot_index += 1
plt.axvspan((Mx.nominal_value - Mx.std_dev), (Mx.nominal_value +
Mx.std_dev), facecolor='g', alpha=0.5)
plt.xlabel(r"$Mass \/ of \/ compact \/ object \/ \/ ( \/ M_\odot \/ )
$")
plt.ylabel(r"$Probability$")
print "Probability > 3 Msol : ", 1 - hist_dist.cdf(3)
print "Mean of Mx : ", Mx_mean
print "Median of Mx : ", Mx_median
print "1-sigma : ", (hist_dist.cdf((Mx.nominal_value + Mx.std_dev))) -
(hist_dist.cdf((Mx.nominal_value - Mx.std_dev)))
plt.show()
```