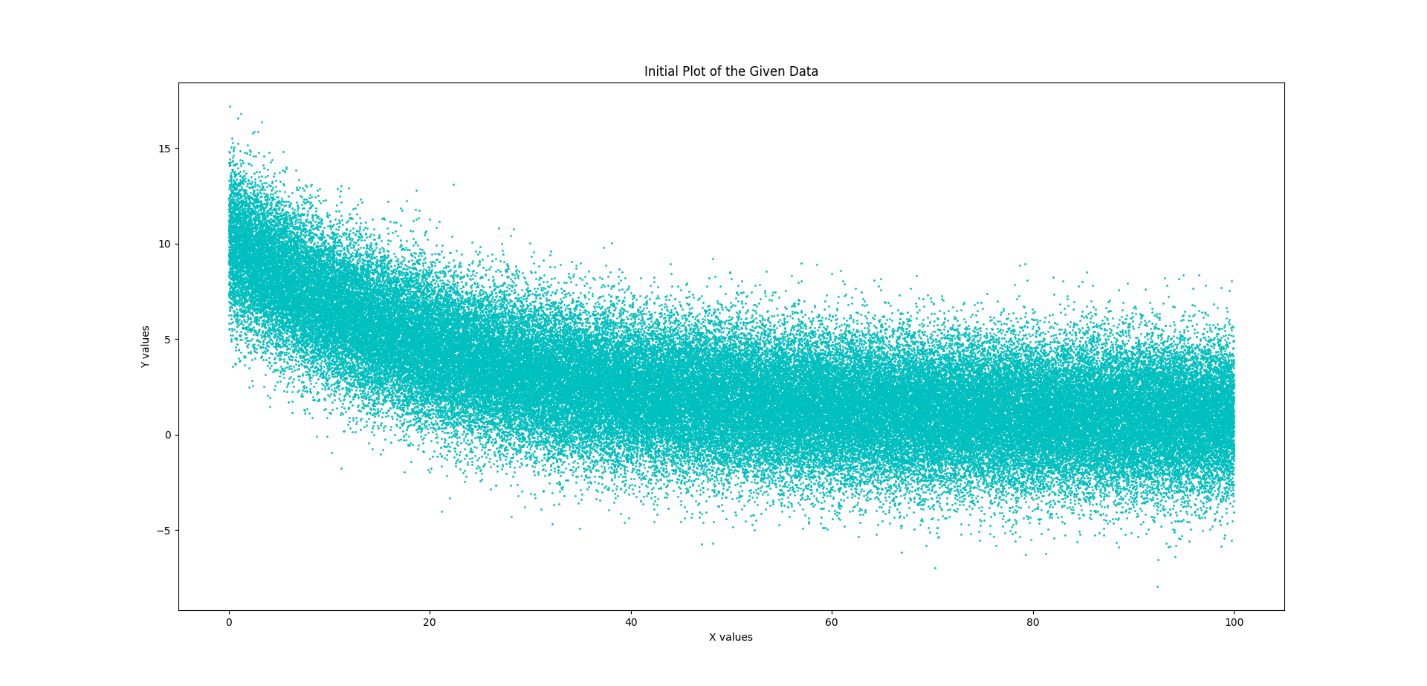
ASTRONOMY 205: HOMEWORK 1

1. Assign1\_1.dat
   1. Plot the data:

**filename = "C:/Users/ryank/Desktop/Work/Classes/Python/ASTR205/Data/"**

**filename += "ASTR 205 1-1.csv"**

**Data = np.loadtxt(filename, comments='#')**

**plt.scatter(Data[:, 0], Data[:, 1], c='c', s=1)**

**plt.title('Initial Plot of the Given Data')**

**plt.xlabel('X values')**

**plt.ylabel('Y values')**

**plt.show()**

Data was imported from the file in its directory using numpy. Then, plotted using a scatter plot from matplotlib and shown.

* 1. Fit using the provided function:

**def Function (X, a, b, c):**

**return a\*np.exp(X/b)+c**

**def chi\_square (fit\_parameters, x, y, sigma):**

**if sigma is None:**

**sigma = 1**

**return np.sum((y-Function(x, \*fit\_parameters))\*\*2/sigma\*\*2)**

**guess = [1, -1, 1]**

**fit\_params, fit\_cov = optimizer(Function, Data[:, 0], Data[:, 1], p0=guess)**

**chi2 = chi\_square(fit\_params, Data[:, 0], Data[:, 1], sigma)**

**dof = len(Data[:, 0]) - len(fit\_params)**

Final Parameters:

A = 9.402e+00 +/- 2.593e-02

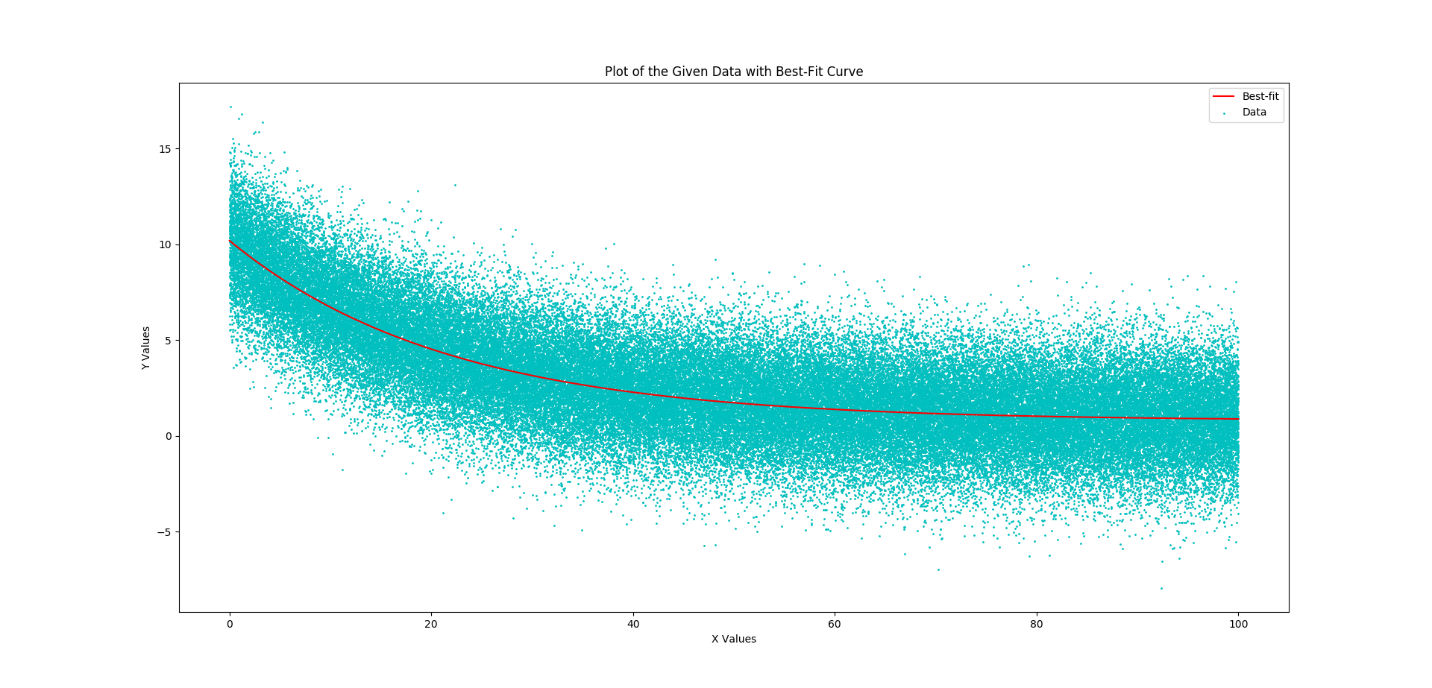
B = -2.169e+01 +/- 1.364e-01

C = 7.960e-01 +/- 1.278e-02

With Chi-squared of 1.1111078192730597

Created two methods for the given function form and calculating the chi-squared of the final parameters. Then a guess is made and put into scipy.optimize.curve\_fit (optimizer) with the function and the data. The parameters and cov are recovered. The chi-squared is then calculated from the parameters and an approximated sigma.

* 1. Plot function using fitted parameters:

**plt.title('Plot of the given data with best-fit curve')**

**plt.scatter(Data[:, 0], Data[:, 1], zorder=1, c='c', label='Data', s=1)**

**plt.plot(Data[:, 0], Function(Data[:, 0], \*fit\_params), zorder=10, c='r',**

**label='Best-fit')**

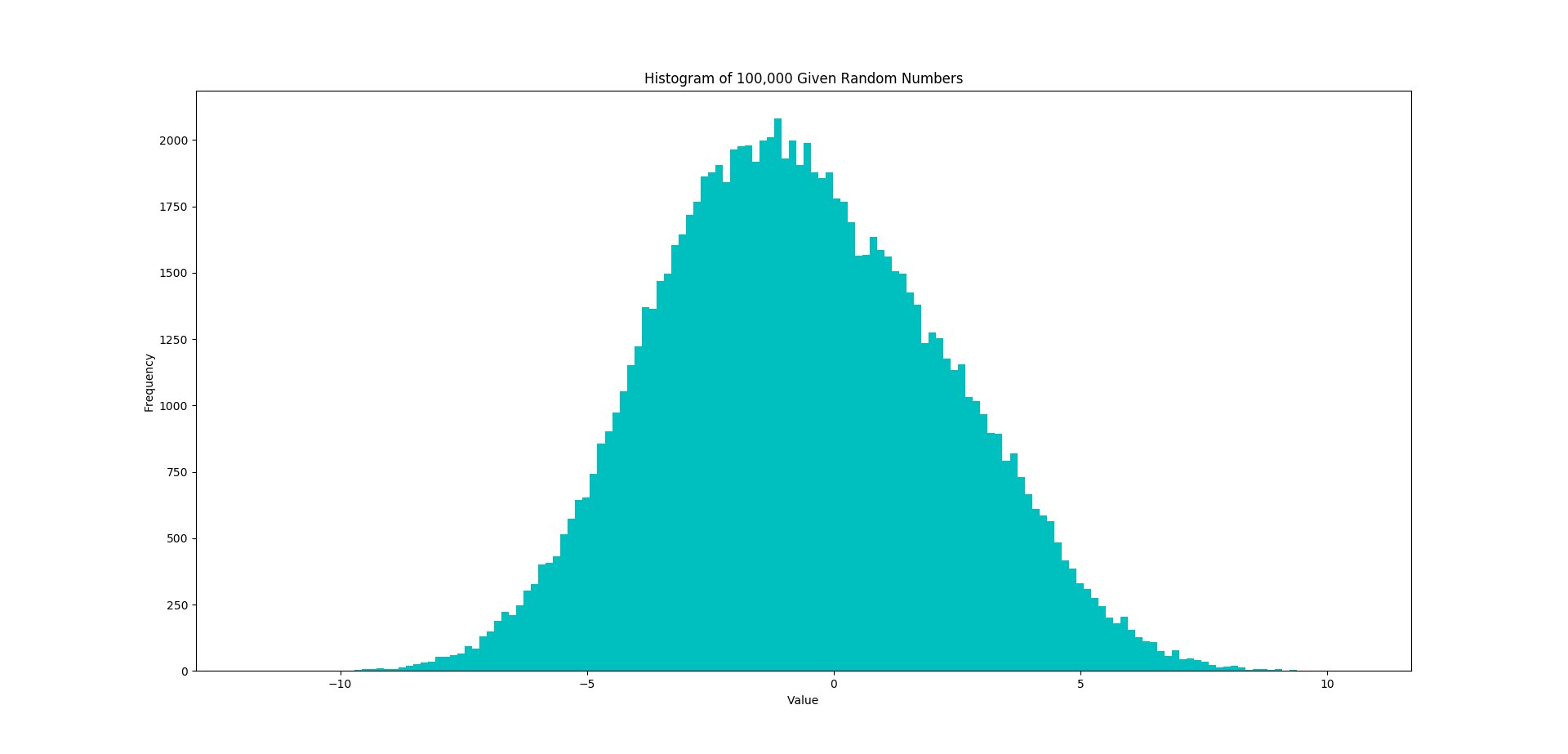
**plt.xlabel('X Values')**

**plt.ylabel('Y Values')**

**plt.legend(loc = 'best')**

**plt.show()**

The fitted function seems very reasonable. As can be seen above, the best-fit seems to go through the middle of all the data. There seems to be an equal amount of points above and below the line. Thus, we conclude that the line is a good fit for the data. The chi-squared backs up this argument, having a value just slightly above 1.

1. Assign1\_2.dat
   1. Create histogram:

**filename = "C:/Users/ryank/Desktop/Work/Classes/Python/ASTR205/Data/ASTR 205 1-2.csv"**

**Data = np.loadtxt(filename, comments='#')**

**Xvals = np.arange(-10, 10, 0.05)**

**alpha = 0.05**

**n, bins, patch = plt.hist(Data, bins=150, color='c', zorder=1)**

**bin\_centers = bins[:-1]+0.5\*(bins[1:]-bins[:-1])**

**plt.title('Histogram of 100,000 Given Random Numbers')**

**plt.xlabel('Value')**

**plt.ylabel('Frequency')**

**plt.show()**

The above is a plot of the given set of 100,000 random numbers. The data was imported with numpy and then plotted using matplotlib. The chosen number of bins is 150. In the above photo, the shape is maintained although it is starting to deteriorate on the right side. 150 seemed the best without compromising the resolution of the image. The distribution is not Gaussian. You can see that the left side does not match the right side.

* 1. Determine if Gaussian:

**W, pvalue = shapiro(Data)**

**print('Shapiro-Wilk Normality Test p-value: {:.2e}.'.format(pvalue))**

**if pvalue<alpha:**

**print('Normality test failed, with {:.2e} < {}.'.format(pvalue, alpha))**

**print('Data is not Gaussian.')**

**print()**

Scipy has a function called Shapiro that performs the Shapiro-Wilk Normality Test on a given data set. The Shapiro-Wilk Normality Test returns a p-value and test statistic W describing how well the data fits a gaussian distribution. The p-value can be compared to a small alpha level to show that the null hypothesis that the data is normally distributed is or is not rejected. If we choose and alpha level of 0.05 (which is standard) and compare to our outputted p-value of 9.01x10-39, we can see that the data does not have a gaussian shape.

* 1. Fit to provided sum of Gaussians:

**initial = [5300,1.3,0,1000,1,-4]**

**plt.hist(Data, bins=150, color='c', zorder=1)**

**plt.plot(bin\_centers, combinedGaussian(bin\_centers, \*initial), 'r--', zorder=10)**

**plt.title('Histogram with Initial Fit for Data')**

**plt.xlabel('Value')**

**plt.ylabel('Frequency')**

**plt.show()**

**plt.hist(Data, bins=150, color='c', zorder=1)**

**params, cov = optimizer(combinedGaussian, bin\_centers, n, p0=initial)**

**plt.plot(bin\_centers, combinedGaussian(bin\_centers, \*params), 'r--', zorder=10)**

**plt.title('Fitted Combined Gaussian for Given Data Values')**

**plt.xlabel('Value')**

**plt.ylabel('Frequency')**

**plt.show()**

Final parameters:

C1 = 2232

Sigma1 = 1.016

Mean1 = 2.188

C2 = 5229

Sigma2 = 1.119

Mean2 = -1.886

The bin centers were extracted from the histogram. Then using scipy.optimize.curve\_fit the combined gaussian is fitted to the given, unnormalized data set. Both graphs were then plotted using matplotlib. The curve\_fit (optimizer) give us the parameters above. C1 and C2 don’t add up to 1, but this is also due to the histograms not being normalized. If we normalize the histograms, we get the following:

C1 = 0.1495

Sigma1 = 1.016

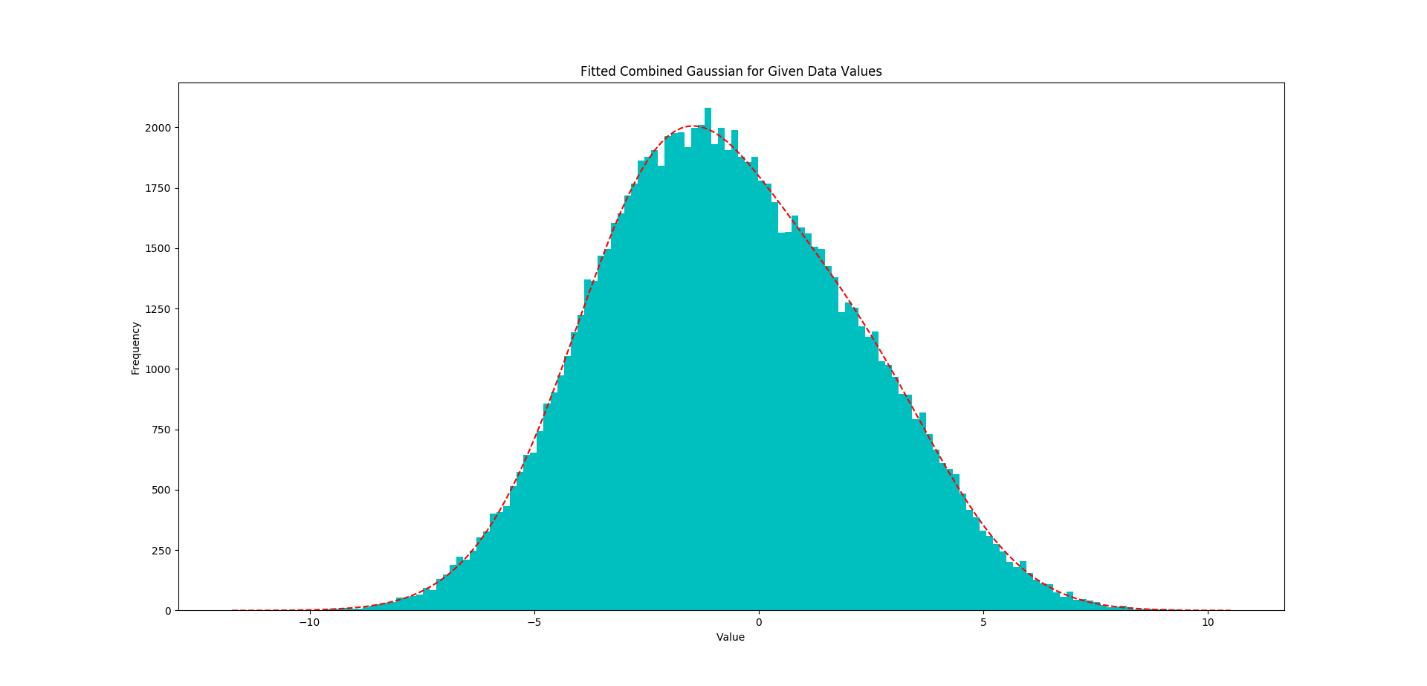
Mean1 = 2.188

C2 = 0.3504

Sigma2 = 1.119

Mean2 = -1.886

The numbers still don’t add up to 1. But they don’t necessarily have to. Each represents how tall their respective curves are. Thus curve 2 is taller than curve 1.

* 1. ****Plot function using fitted parameters:

The above plot is the data with the given combined gaussian fit. From the graph, it seems that the curve-fitted graph matches well with the data that we were given. The data and curve were again plotted with matplotlib using the parameters received from the scipy.optimize.curve\_fit function.