# Assignment 2

Student: 6463360

*Department of Physics, University of Surrey, Guildford GU2 7XH, United Kingdom*

Three problems have been approached using techniques in Python. The first was addressing the effect of diffraction on a telescopes resolution. Using a specially written Python programme a diffraction pattern of light from an infinitely distant source was simulated. This was done by writing a function that calculates a Bessel function. The function was tested against SciPy's Bessel calculator and was found to be accurate. The resulting intensity distribution (diffraction pattern) produced by the programme provides insight into how various factors affect the resolution of a telescope. The second problem considered the process of generating a random and discrete sample of data that resembles a parent distribution. This was done using the transformation method. The programme implemented this technique well and the resulting graphs showed an appropriate fit of the data to the parent distribution. This practice can prove useful for comparing observed data to data from a distribution that it is thought to fit. The final problem explores fitting data to a distribution. By assuming the general nature of a data set, one can use the method of maximum likelihood to calculate the variables in the general form of the PDF that best describes the data. This method proved to be successful and the programme written was efficient in calculating the variables that define the PDF. The values of $\alpha$ predicted followed a Gaussian distribution. 67.9% of the calculated $\alpha$ values were within $1\sigma$ of the correct value.

## 1. Simulating the Diffraction Limit of a Telescope Using Numerical Integration

### A. Introduction

Bessel functions appear in many areas of physics. There are Bessel functions as coefficients in the calculations of the perturbations of planets[1]. There is a Bessel function in the calculation of the light intensity of light in a diffraction pattern, too.

$$I(r) = \left(\frac{J_1(kr)}{kr}\right)^2 \qquad (eq.1)$$

$$J_1(x) = \frac{1}{\pi}\int_0^{\pi} \cos(\theta - xsin\theta)\, d\theta \qquad (eq.2)$$

In equation 1 the Bessel function is defined by equation 2 for $x = kr = \frac{2\pi}{\lambda}r$.

When observing light through a telescope, it is diffracted and forms circular rings of light and dark regions. The size of the central most intensity disc determines a telescopes resolution[2]. By simulating the diffraction pattern one can determine for which wavelengths a telescope will be most effective and if there will be shortcomings in its observations.

Simpson's rule is used in mathematics for approximating the integral of a given function, using discrete data points[3]. This proves useful when integrating sets of data in a programme where an infinite number of points (to make the function continuous) is not possible. For example, producing a plot of a Bessel function.

This section has determined that equation 2 can accurately be calculated by creating a programmatical function in Python. There are programmatical libraries that are able to calculate integrals using techniques such as Simpson's rule. For example, SciPy. This was utilised in the function when calculating equation 2. A comparison between the written $J_1(x)$ function and SciPy's version of the function has been made to show this.

A diffraction pattern can be simulated for inspection. This can be useful such that scientists can determine under what parameters a telescope may or may not be efficient. Equation 1 has been solved by the programme to produce an image of the diffraction pattern described.

### B. Methods

The first job was to make a Python function that calculates equation 2. The integration was done by utilising the SciPy Simpsons rule function. The SciPy library was imported for this. Numpy was also imported for use with trigonometric functions defining $\pi$, among other things. The integration was done over the range of 0 to $\pi$ with 1000 points. The function – dubbed MyJ1 – was then used to calculate the Bessel function for 1000

values of x over the range $0 \leq x \leq 20$. The same was done using the SciPy Bessel function and both were plotted on a graph for comparison.

The diffraction pattern was then calculated for a wavelength of $\lambda = 500nm$. The radius of the diffraction pattern on the focal plane was taken as $r = 1\mu m$. Intensities at 500 points along this r were calculated in 2 dimensions. This resulting pattern was then copied, rotated and stitched together to create a small mosaic that showed the whole pattern over the whole plane of diameter 2r. To prevent the central most intensity spot from causing too much glare on the image, a limit of 0.01 was made for the intensity. This altered the rest of the intensities at greater values of r to be such that they were relative to 0.01 maximum intensity.

Both the plot and image were saved as images by the programme when it ran.

### C. Results

The programme produced a graph that showed the MyJ1 function and SciPy's $J_1(x)$ function for comparison. This can be seen in figure 1. As the functions are graphically identical the black line was thinned and the blue one segmented into dots to allow them both to be seen on the same graph.

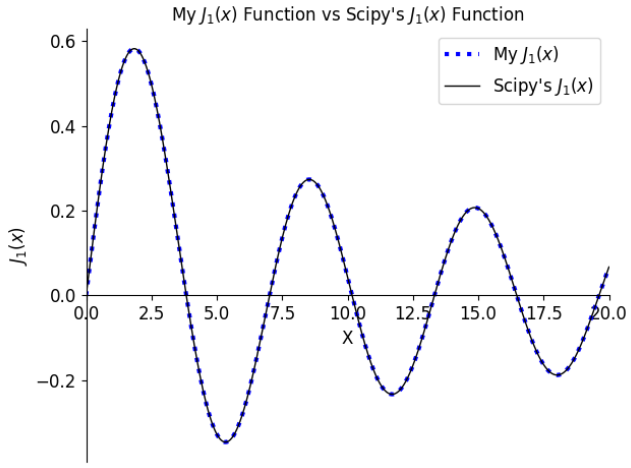


Figure 1 A plot demonstrating the accuracy of MyJ1 Bessel function by comparing it with the SciPy libraries integrated Bessel function calculator.

The diffraction pattern produced by the programme can be seen in figure 2. By using the units stated on the image it can be seen that the central circle is approximately $0.4\mu m$. The grayscale and maximum intensity was adjusted in the programme such that the image produced best resembled the example in the assignment brief.

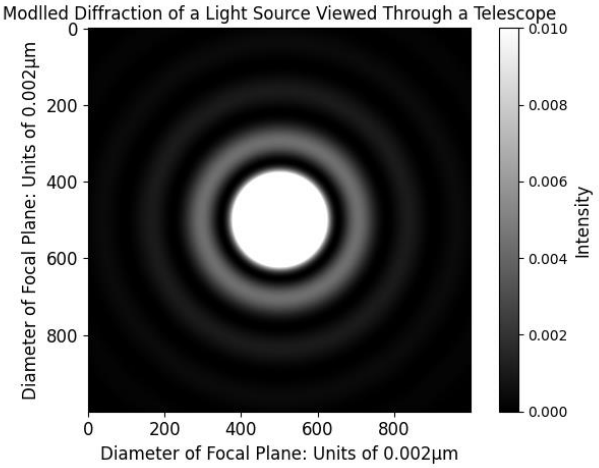


Figure 2 The simulated diffraction pattern of light on a telescope from a point source. The light had a wavelength of $\boldsymbol{\lambda = 500nm}$.

### D. Discussion

It is clear from an examination of figure 1 that the Python function, MyJ1, produces an identical result for the Bessel function as the SciPy $J_1(x)$ function. The amplitude reduces as x increases and will tend towards 0 as x tends to infinity. Although the amplitude may decrease as x increases, the period of oscillation remains constant, as can be seen in figure 1.

The retardation of the oscillating wave that describes the Bessel function resembles the intensity decreasing as r increases, while retaining a constant period of oscillation. It is clear that the Bessel function describes the general wavelike property of the intensity diffraction pattern. When $J_1(x) < 0$ in figure 1, the rings are dark in figure 2. Similarly, when $J_1(x) > 0$, the intensity rings exhibit constructive wave interference. The diffraction pattern matches the diffraction pattern of the same description from the book *Computational Physics*[4]. This validates the results produced by the programme.

Being able to make these simulations is useful to the design specification of telescopes. By altering the radius of the focal plane and the wavelength of the light source, parameters can be calculated that best fit a particular telescopes purpose. If there is more than one light source, it is required to know whether they will both be

visible on a telescopes plane. If the first black ring of one abject encroaches on the central intensity disc of another object, the objects are considered to be blurred together.

### E. Conclusions

The Python programme was able to successfully calculate the Bessel function to a degrees as accurate as the SciPy function. This was then effectively used for computing equation 1. The programme generated an image that simulates the diffraction pattern of light at $\lambda = 500nm$.

By comparing figures 1 and 2, sensible conclusions were drawn about the role the Bessel function plays in describing the diffraction pattern.

The programme ran efficiently, although the time taken to produce figure 2 could be reduced. Coding a way of creating the pattern without stitching multiple images together would be an improvement on the current version.

## 2. Randomly Generating a Discrete Sample from a Probability Density Function

### A. Introduction

There are multiple reasons why one may generate seemingly random data sets from a parent distribution; it can be useful to generate data from a PDF that we believe our data resembles and compare the data, calculating the initial conditions of a system can be done by analysing generated data from a PDF, errors can be calculated for the randomly generated data set and compared to the errors in our observed data to better understand it.

A useful technique of generating random data that resembles the parent distribution is the Transformation method[5]. In this method, the PDF, f(x), is first normalised. Then the CDF, F(x), and inverse CDF, $F^{-1}(R)$, are found respectively. This allows for a random datapoint to be calculated by calculating the function $F^{-1}(R)$ for a random number, R, in the range $0 \leq R \leq 1$. The resulting datapoint falls within the bounds of the original PDF and lies in the range $xlo \leq x \leq xup$ which is the original range of the PDF.

This section deals with generating a random data set from the initial parent PDF. 2 histograms are produced for each value of $\alpha$ that is programmed. The programme runs for values of $\alpha = -2.35, -1.00$. An analysis of errors is also addressed in this section.

### B. Methods

Prior to writing the Python programme, some calculations were required. The transformation method involved using the probability density function (PDF), f(x), the cumulative distribution function (CDF), F(x), and the inverse of the CDF, $F^{-1}(x)$. The PDF also needed to be normalised. These calculations are described in the following steps. The initial PDF given with the range $xlo \leq x \leq xup$ was:

$$f(x) = Ax^{\alpha} \qquad (eq.3)$$

A python function for equation 3 was defined in the programme such that it could be used to plot the parent distribution for comparison with the randomised data set.

1. Normalising the PDF.

In order to normalise the function and calculate the constant, A, f(x) was integrated over the whole x range and set equal to 1.

$$\int_{xlo}^{xup} Ax^{\alpha} dx = 1 \qquad (eq.4)$$

Solving this and rearranging for A gave equation 5. As can be seen, if $\alpha = -1$ there would be a division by zero which is impossible. A special case was required for $\alpha = -1$. This was programmed into a Python function called calcA in which an if statement decided which calculation to use based on the value of $\alpha$.

$$A = \begin{cases} \alpha \neq -1 & \dfrac{\alpha + 1}{x^{\alpha+1} - xlo^{\alpha+1}} \\ \alpha = -1 & \dfrac{1}{\ln|xup| - \ln|xlo|} \end{cases} \qquad (eq.5)$$

2. Calculating the CDF.

The CDF was calculated by integrating the PDF up to a certain value of x. In the case of the transformation method the complete CDF was required. This meant that the x used was xup. A special case was required for $\alpha = -1$ as was the case for the PDF.

$$F(x) = \begin{cases} \alpha \neq -1 & A(\ln|x| - \ln|xlo|) \\ \alpha = 1 & \dfrac{A}{(\alpha+1)(x^{\alpha+1} - xlo^{\alpha+1})} \end{cases} \qquad (eq.6)$$

The CDF was programmed as a Python function called CDF. A characteristic of the CDF is that $0 \leq F(x) \leq 1$ for the entire x range $xlo \leq x \leq xup$. A test for this was programmed with an exit statement to end the programme with an error message if this condition was not met.

3. Inverting the CDF.

The CDF was set equal to R such that $F(x) = R$ for $0 \leq R \leq 1$. The CDF was then inverted to allow for a calculation of x. The equation for this follows.

$$F^{-1}(R) = x$$

$$x = \begin{cases} \alpha \neq -1 & e^{R/A + \ln|xlo|} \\ \alpha = -1 & \left(\dfrac{(\alpha+1)R}{A} + xlo^{\alpha+1}\right)^{1/\alpha+1} \end{cases} \qquad (eq.7)$$

A Python function was written to calculate equation 7. The randomised data set was generated by running this function with 10,000 random values of $0 \leq R \leq 1$.

4. Running the programme.

Taking the two values $\alpha = -2.35, -1.00$, the corresponding values of the normalisation constant, A, were calculated and printed to the user. The programme has been written in such a way that loops were utilised to make these calculations. The $\alpha$ values were contained in a list. The range of these loops was the $\alpha$ list. This allowed for the user to change the values and even the quantity of $\alpha$ values without the programme failing. The programme was able produce 2 histograms for any range and number of $\alpha$ values with only the variables needing to be changed.

Once the A values were calculated the programme checked the F(x) condition. If the condition was not met, the programme would have ended and printed a custom error to the screen. It then moved onto creating the randomised data set and plotting the 2 different types of histogram. This ran in a loop whose range is the $\alpha$ list, much like the previous loop for the same reasoning described.

With each run of the loop, a randomised data set was generated and sorted. This sorted data set was used to create a list of bin values. A value called logbins was created which describes to the programme how to bin the data for a log graph. For the normal histogram the bin list provided the number of bins the user wants to generate. (In this case nbins=100.) In the case of the log graph, the bin list provided the logbins value.

The next step in the programme was to plot the histogram. This was done with the appropriate bins. Then the error bars were calculated and added to the plot. The errors were calculated by assuming that each bin in the histogram could be considered to be a Poisson process. This allowed for the standard deviation to be approximated as:

$$yerr = \frac{\sqrt{bin\ counts}}{bin\ counts} \qquad (eq.8)$$

Once the error bars were applied, some plot formatting was done and the figures were saved as well as being shown to the user.

### C. Results

Each value of $\alpha$ coincided with 2 graphs being generated. Figures 3 and 4 are the linear and logarithm graphs for $\alpha = -2.35$ respectively. Figures 5 and 6 are show the $\alpha = -1.00$ graphs. No error was recorded while running the programme so it is safe to assume that the CDF met the requirements set out in the previous section. When the programme ran, the following values for the constant A were printed to the screen:

$$\alpha_0 = -2.35, A_0 = 1.523$$
$$\alpha_1 = -1.00, A_1 = 0.621$$

These values can be seen in the graphs produced in the form of the function, f(x).

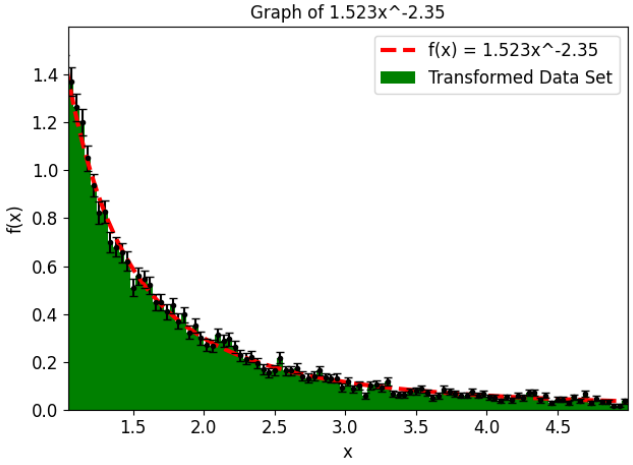


Figure 3 Shows the random data generated for $\boldsymbol{\alpha = -2.35}$ by the programme with the parent distribution plotted to demonstrate how well it fits.

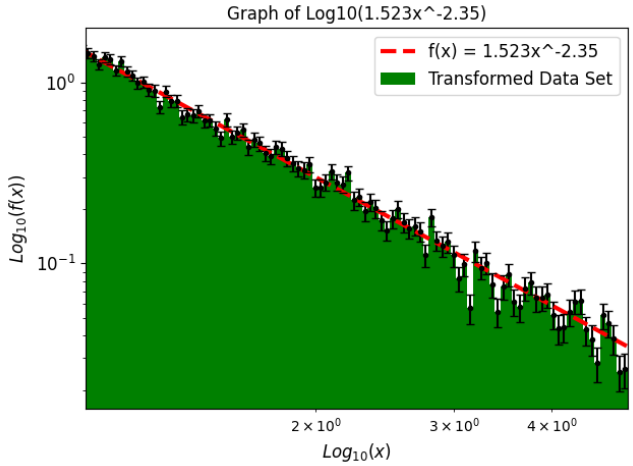


Figure 4 Shows the data from $\boldsymbol{\alpha = -2.35}$ logged. The log of the parent distribution has also been plotted to demonstrate the fit.

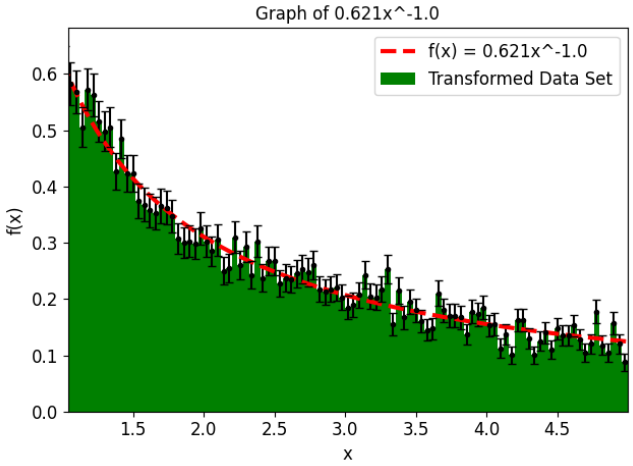


Figure 5 Shows the random data generated for $\boldsymbol{\alpha = -1.00}$ by the programme with the parent distribution plotted to demonstrate how well it fits.

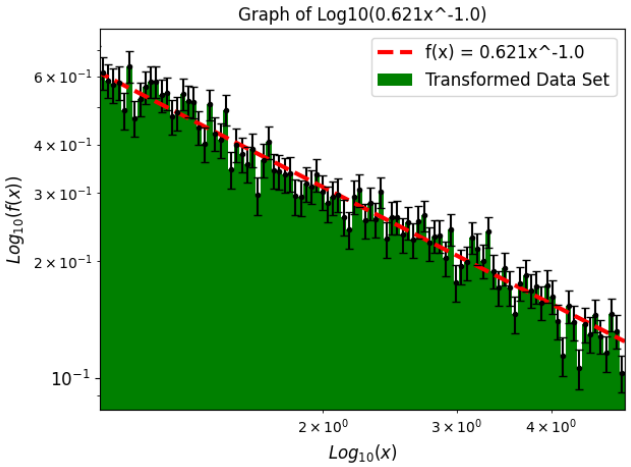


Figure 6 Shows the data from $\boldsymbol{\alpha = -1.00}$ logged. The log of the parent distribution has also been plotted to demonstrate the fit.

## D. Discussion

Figures 3 and 5 illustrate that the transformed data fits the parent PDF well. The error bars appear to reduce in size as x becomes larger. The majority of bins appear to fall within an error range of the PDF - plotted as the red line.

Figures 4 and 6 show the logged data. Logging the PDF makes it a linear equation. (As shown in equation 9.) The data fits well to the parent distribution in these graphs, too. The error bars appear to be larger for increasing values of x and smaller for smaller values of x. This is the opposite of what was seen in figure 3.

$$\ln\big(f(x)\big) = \ln(A) + \alpha \ln(x) \qquad (eq.9)$$

The data fitting well to the parent PDF was expected. The transformation method generates data that follows the distribution when done correctly[5]. This validates that the method was implemented well.

## E. Conclusions

The transformation method for generating random data that resembles a parent PDF works well. The Python programme was successful and due to the looped nature of its structure it can be used to test more values of $\alpha$ with ease. Having a linear relationship such as equation 9 can be useful as will be shown in section 3, equation 10. Being able to generate linear data on log graphs can aid in this.

Research into different ways of generating random numbers may improve the programme by producing more random results. This would allow it to resemble an observed data set more closely.

## 3. Calculating Likely Values of $\alpha$ From Discrete Data

### A. Introduction

Fitting data to a distribution is an important aspect to statistical physics. Often a data set will exhibit a shape that can be assumed to be of a particular form, e.g. linear, power law, etc. By using the method of maximum likelihood one can use a data set and the general distribution equation it resembles to determine the equation that best describes the data[6].

In this section random data were generated using the methods from the previous section. This was then assumed to follow a power law distribution and the methods of maximum likelihood were used to find the most likely value of its variables.

### B. Methods

To generate the random data set the programme utilised the methods from the previous section. The assumed distribution that the data fit was equivalent to equation 3. Using the following equations from the book *Data Reduction and Error Analysis for the Physical Sciences*[6] functions were written in Python to calculate the log likelihood of a range of $\alpha$ values, $-4 \leq \alpha \leq 0$.

$$M = \sum_{i=1}^{N} \ln P_i \qquad (eq. 10)$$

$$P_i = f(x_i) = Ax_i^{\alpha} \qquad (eq. 11)$$

Equation 10 gave a value for how accurately the value of $\alpha$ fit to the data. The higher the value of M, the more likely the value of $\alpha$ was the correct variable. The variable A was calculated based on this $\alpha$ value, too. This was done using the function created from equation 5. The programme calculated M for the range of $\alpha$ values and saved them to a list. This list was plotted in the first run of the programme and saved.

In order to find the $\alpha$ value that best described the random data, the M was flipped by calculating the minus log likelihood. This made it possible to use the SciPy Optimize library to find the minimum value of the curve. This provides the most likely value of $\alpha$ from the likelihood equation (equation 10).

To calculate the error in this most likely value, the following equation was used:

$$M(\alpha \pm \sigma_\alpha) = M_{max} - 0.5 \qquad (eq. 12)$$

By approximating the log likelihood curve as a Gaussian, one can use equation 12 to calculate the error, $\sigma_\alpha$, in the most likely value of $\alpha$[6]. A function was written to return $M - M_{max} + 0.5$. Using the SciPy Optimize library, this function was run through the bisect function to find the point at which it equals zero. This process allowed for the calculation of $\sigma_\alpha$. These $\sigma_\alpha$ values were saved into a list for use later.

The programme calculated if the most likely $\alpha$ value was within $1\sigma_\alpha$ of the actual value, $\alpha = -2.35$. The values that satisfied this were added to an accept list and the ones that didn't were added to a reject list. The accept list was used to print the percentage of $\alpha$ values that were within $1\sigma_\alpha$ of $\alpha = -2.35$.

This process was looped 1000 times and each time, a new set of random data was generated. The resulting $\alpha$ values were saved into a list and the programme created a histogram of the most likely $\alpha$ from them. The histogram was saved to an image file and printed to the screen for the user to see.

### C. Results

The programme outputted the x range that the data were in, $xlo \leq x \leq xup \rightarrow 1.00004 \leq x \leq 4.99683$. The percentage of $\alpha$ withing $1\sigma_\alpha$ of the actual $\alpha = -2.35$ was printed as 67.9%. The programme saved and showed the graphs in figures 7 and 8 upon completion.

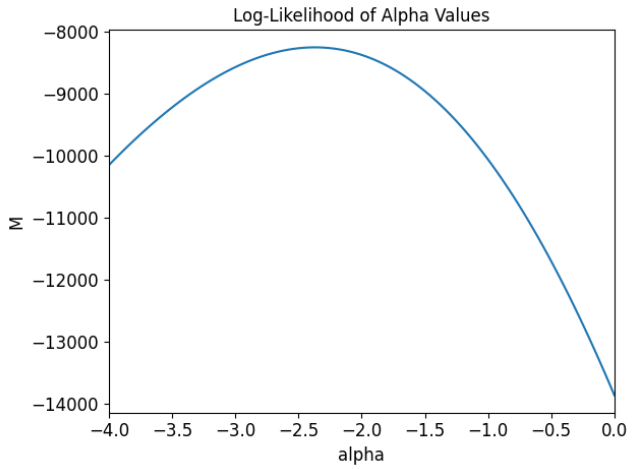


Figure 7 The log-likelihood of a range of $\boldsymbol{\alpha}$ values for a random set of data. The peak of the curve corresponds to the most likely value of $\boldsymbol{\alpha}$ for this distribution.

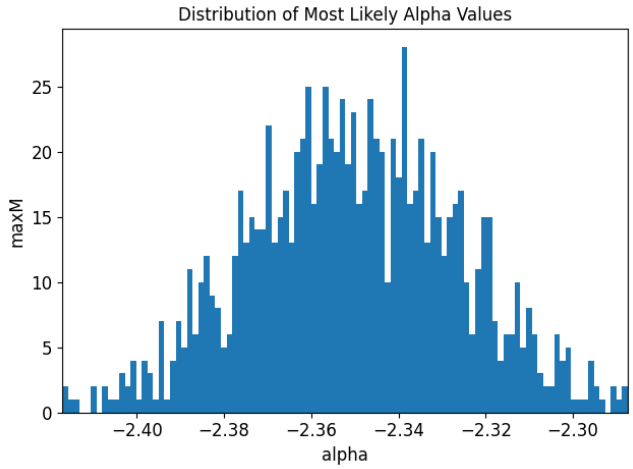


Figure 8 The distribution of the most likely value for $\boldsymbol{\alpha}$ for 1000 random data distributions.

## D. Discussion

The values of xlo and xup were as expected. When generating a data set numerically, there is no certainty to getting the x limits to be exactly as expected. These values are very close to the x limits used to generate the data. The x limit values for each set of data was used to calculate that data's A coefficient. The percentage printed for the number of $\alpha$ values within $1\sigma$ of $\alpha = -2.35$ was very close to 68% which is the number of samples within $1\sigma$ of the mean of a perfect Gaussian curve[7]. Upon examining figure 8, it is clear that the distribution of the most likely $\alpha$ values resembles a Gaussian distribution. The percentage output by the programme further confirms this.

By looking at Figure 7 one can assume that the maximum value corresponds to an $\alpha$ value between -2.5 and -2.0. This is as expected as the real value is $\alpha = -2.35$. The programme calculated the actual value using the methods described previously. The range of $\alpha$ values is shown on the axis of figure 8. It agrees with what can be seen in figure 7.

## E. Conclusions

The values of xlo and xup were correct as these values were used to define the random data to begin with. Therefor the programme ran correctly in creating the randomised data.

The log likelihood graph in figure 7 looked as expected[6]. There was a distribution of $\alpha$ values with a peak M that corresponds to a value close to the correct $\alpha$. The histogram in figure 8 clearly shows the most likely value of $\alpha$ being close to the correct value and is within $1\sigma$ of the correct value. This leads to the conclusion that the method of likelihood fitting is effective in finding the correct parameters that define the PDF for the data. To ensure an accurate value for $\alpha$ is calculated it is recommended that the programme ran at least 1000 times (as it did to create the figures). This ensured the histogram in figure 8 looked more like a smooth Gaussian curve. This was beneficial because it allowed for a more accurate representation of the most likely value of $\alpha$.

The programme could be improved by repeating the $\alpha$ estimation more than the 10,000 times used in the current version of the programme. This would provide a much smoother plot of figure 8. However, the run time would increase drastically.

## References

1. **Fricke, Walter.** Friedrich Wilhelm Bessel . [Online] Encyclopedia.com, 2008. [Cited: 11 22, 2020.] https://mathshistory.st-andrews.ac.uk/DSB/Bessel.pdf.

2. *On the Diffraction of an Object-glass with Circular Aperature.* **George Biddell Airy, A.M.** s.l. : Transactions of the Cambridge Philosophical Society, 1834, Vol. 5, pp. 283-291.

3. **Grewal, B. S.** *Numerical Methds in Engineering and Science.* New Delhi : Mercury Learning and Information, 2019.

4. **Newman, M. E. J.** *Computational Physics.* s.l. : CreateSpace Independent Publishing, 2013. p. 148.

5. **Press, William H., et al.** *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge : Cambridge University Press, 2002.

6. **Robinson, D. Keith and Bevington, Philip R.** *Data Reduction and Error Analysis for the Physics Sciences.* s.l. : McGraw-Hill, 2003.

7. **Glen, Stephanie.** Symmetric Distribution in Statistics. *StatisticsHowTo.com.* [Online] September 18, 2013. [Cited: December 27, 2020.] https://www.statisticshowto.com/symmetric-distribution-2/.