Optics with Imperfect Mirrors

Head of class - David Buscher

April 23, 2016

Abstract

Diffraction patterns (i.e. the Point Spread Functions (PSF)) of various apertures were generated numerically using a Fast Fourier Transform in the Python language. The effect of a 'tapered' aperture function where $A(r) \propto e^{-r^2/\sigma_T^2}$ was modelled - it was found that the on-axis intensity fell as $I \propto (\sigma_T/R)^4$, whilst the Full Width at Half Maximum amplitude (FWHM) went as $W \propto (\sigma_T/R)^{-1}$. Qualitative analysis of the effect of a central hole in the mirror was carried out and shown in figure 7. Analysis of independent Gaussian-distributed errors with an rms value σ_ε at each point in the aperture showed that the on-axis intensity was steady for $\sigma_\varepsilon/\lambda$ less than 0.1, at which point it dropped by a factor of 10^4. A qualitative analysis of extended errors with a Gaussian profile was carried out and is shown in figure 10.

1 Introduction

It has been appreciated for over 150 years that the wave nature of light leads to definite limitations on the performance of telescopes, even in the absence of any mechanical defects on the lens or mirror itself. Airy [1] was the first to develop a full quantitative treatment of the optical behaviour of a lens. His analysis is still relevant today, although the state of the art has moved on significantly. Today's optical telescopes record directly onto electronic detectors, the data from which is fed into sophisticated image processing software, using numerical techniques to determine the true patterns of light incident on the apertures. In this project, we examine the diffraction pattern of a 6-metre-radius telescope, first treating the telescope without any errors before moving to incorporate errors in the telescope itself. The analysis was performed using numerical techniques, relying heavily on the Fast Fourier Transform (FFT) [2] using the numpy and scipy libraries in python.

2 Theory

It is well-known that the far-field diffraction pattern of an aperture may be obtained by taking the Fourier transform of the aperture function. In astronomy the diffraction pattern obtained is known as the Point Spread Function (PSF) [3], and is understood to show the pattern of light produced when a δ-function source of light illuminates the telescope. If we let the aperture function be denoted as $A(x, y)$, where x and y are directions in the aperture plane, we can represent this as

$$\text{PSF} = \tilde{A}(\phi, \theta) = \text{FT}(A(x, y)), \tag{1}$$

where ϕ and θ are the directions in the angular pattern of light from the telescope.

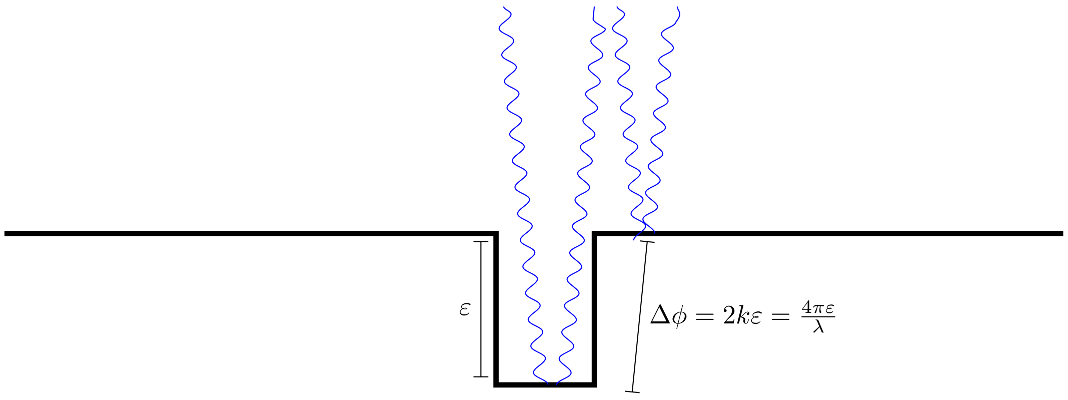
ε
$\Delta\phi = 2k\varepsilon = \frac{4\pi\varepsilon}{\lambda}$

Figure 1: Showing the change in phase due to a physical distortion in the mirror surface, modelled here as a hole of extent ε. The phase change developed is $\frac{4\pi\varepsilon}{\lambda}$.

For a perfectly circular mirror of radius R, we would expect the complex aperture function to be simply

$$A(x,y) = \begin{cases} 1 & \text{if } \sqrt{x^2+y^2} < R \\ 0 & \text{if } \sqrt{x^2+y^2} > R \end{cases} \tag{2}$$

However for aperture functions that model more realistic telescopes, we can have a variety of effects rendering the aperture function more complicated. In particular we may have a tapered aperture function, such that A is given by

$$A(x,y) = \begin{cases} \exp\left(-(x^2+y^2)/2\sigma_T^2\right) & \text{if } \sqrt{x^2+y^2} < R \\ 0 & \text{if } \sqrt{x^2+y^2} > R \end{cases} \tag{3}$$

Such a tapered aperture function is often incorporated into radio telescopes by design. However often telescopes develop complicated aperture functions due to errors in the telescope itself. It is easy to see how hard it is to ensure that a 6 metre radius telescope is perfectly flat, particularly when it must be held up under its own weight and exposed to dust and other contaminants from the atmosphere. In particular, when dealing with a physical indentation in the telescope of depth ε, figure 1 shows that this will lead to a phase error of $4\pi\varepsilon/\lambda$. Perhaps the most dramatic example comes from the Hubble Space Telescope, where a 2.2 μm error in manufacturing led to years of significantly reduced focussing ability and cost NASA tens to hundreds of millions of dollars to fix [4].

3 Method

3.1 Overview

The python programming language was chosen for this task, due to the easy availability of numerical and scientific libraries, in addition to many plotting tools. Python is an interpreted language

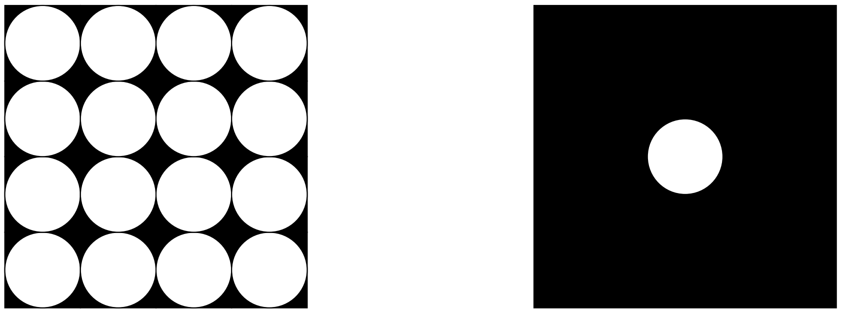

Figure 2: Showing the difference between circular aperture functions that do and do not have padding. On the left is the result when the FFT algorithm tiles an infinite plane with a non-padded circular aperture. On the right is the same section of the tiled plane if the aperture function is padded significantly. On the right the aperture function represents a solitary mirror much more accurately.

however, which typically leads to large performance reductions compared to a compiled language. By performing most operations in calls to the numpy library, which runs off compiled routines, this overhead can be substantially reduced, and this was an approach I heavily relied on in this exploration.

Broadly speaking, the computational framework was straightforward - the various aperture functions, which were simply arrays of complex numbers, had to be generated and then an FFT library was used to take their Fourier transform. The resulting transform was the diffraction pattern, i.e. the Point Spread Function. The FFT routine chosen was `numpy.fft.fft2d`, which gives a 2-dimensional Fourier transform, and is a very well-tested routine from an established library. Instead of choosing a fixed screen distance and representing the extent of the pattern in terms of distances along this screen, it was decided to take a more general approach and represent the extent of the pattern with angles, understanding that the pattern shown is only valid for the Fraunhofer regime.

One physical problem that was encountered was the need for a sufficient amount of padding. The FFT routine works on the assumption that the pattern supplied is repeated infinitely in every direction. With no padding, the routine is supplied with an aperture function consisting of a transparent disc inscribed in an opaque square. However when this is repeated infinitely this is very similar to the pattern of an infinite transparent plane, with only 20% of the area opaque. This is shown in figure 2. To properly represent the aperture function consisting of a disc in an infinite plane of non-transmitting substance, it's necessary to pad the pattern with zeros. This gives the correct diffraction pattern. Padding with zeros also increases the resolution of the diffraction pattern, reducing pixellation in the output. Along with the amount of padding, the number of pixels used to represent each square metre in the aperture could be varied too. Typically between 100 and 400 pixels were used for each square metre in the aperture, as it was found that increasing the resolution beyond this didn't result in significantly increased accuracy yet took much longer to compute.

The output of the Fourier transform must be scaled correctly, and so it's important to work out the scaling factor. This is the angular displacement corresponding to traversing one pixel in the diffraction pattern array. Using [5] it's fairly easy to show that the scaling factor $\gamma = \lambda/(N * \Delta)$, where Δ is the difference in the distance between adjacent cells in the aperture array. Since the diffraction array generated was very large, generally only the angular spread corresponding to the central few minima was considered, as typically only the near-axis waves will be captured by a CCD, human eye or radio antenna.

An object-orientated approach was taken to organise the code neatly, with a Mirror class containing data and methods. The data were two arrays of complex numbers, the aperture array and the pattern array. The aperture function was generated when the object was instantiated, and could be updated with various object methods which could add a central hole, add point errors, or add large-sale-errors. The pattern was generated with a method that a `Mirror` object could call . This method could update the pattern array after the aperture function array had been modified to reflect the new diffraction pattern that would be formed.

3.2 Implementation

The aperture array was a numpy array of type `complex`. A function `taper` was created to generate a square array with a gaussian amplitude profile. On instantiation of the Mirror object, an array of side `2R * points_per_m` was created with the tapered aperture. Then array cells greater than one radius from the centre of the array were set to 0. This was then embedded in an array of zeros of side $n =$ `2 * R * points_per_m * pad_factor` to achieve the necessary padding.

A utility function `find_width` was created to help calculate the Full Width at Half Maximum (FWHM) amplitudes. This assumed a symmetric diffraction pattern, so was less useful for the more heavily distorted asymmetric patterns.

The pattern was calculated by simply invoking numpy's 2D FFT routine, `fft2d`. This was then shifted to the correct orientation for pattern visualisation using `fftshift`. The scaling factor was set as described in section 3.1.

A hole was set in the centre of the mirror by setting points greater than the supplied radius to 0, very similarly to how the circular aperture was created above.

Adding point errors was more involved. Here, every point in the aperture gained a phase $4\pi\varepsilon/\lambda$, where each ε is individually and independently normally distributed with an rms value σ_ε. Numpy made this fairly easy, as an array of gaussian-distributed numbers ϵ could be created and an elementwise complex exponential taken using `np.exp` to form an array of $\exp(4i\pi\varepsilon/\lambda)$. This phase array could then elementwise multiply the complex aperture array, automatically only affecting those points with non-zero aperture function value.

The final functionality of the Mirror class was adding large-scale errors to the aperture array, i.e. adding gaussian-profile extended phase errors with correlation length ℓ_c. This was achieved by generating a random (x,y) pair uniformly over the rectangular set of (x,y) values that the circular part of the aperture was inscribed within. Then a $3\ell_c$ by $3\ell_c$ square array was created using the `taper` function described above. This cutoff was chosen because of simplicity and the fact that at $3\ell_c$ the value of the gaussian is less than 0.1% of the value at the centre. The phase error was calculated from this array, and then the aperture array multiplied by this array, centred at the (x,y) coordinates calculated above.

3.3 Efficiency and Errors

In order to perform the computational investigations in a reasonable amount of time, it's necessary to be able to carry out the process of generating an aperture and calculating the diffraction pattern in a relatively short period. Of course, spending extra time and memory working to more precision

Point Errors	__init__	Pattern	Large Deformations	Add Hole
0.64	0.0087	0.53	0.068	0.048

Figure 3: Typical times in seconds spent inside each subroutine for a single instantiation of a `Mirror`, followed by adding a hole, adding point errors, adding large-scale errors and taking the diffraction pattern. Times generated using `pycallgraph` and $n = 2,400$, on a 2.9 GHz Intel Core i5.

than is needed is wasteful. As discussed in section 3.4 the main source of errors in the result was quantisation of the diffraction pattern. This can obviously be reduced by increasing the amount of padding, and it was found that a padding factor of ten or twenty was sufficient to reduce the error to acceptable levels.

Similarly, increasing the resolution of the aperture function decreased the quantisation error there too. It was found that a resolution of 100 points per square m was sufficient. `Pycallgraph` [6] was used to time and visualise the running time of the various subroutines, under typical conditions of 100 points per m^2, a 6m radius, and a padding factor of 20, so that the aperture array was of dimensions 2,400 $\times$ 2,400. Results can be seen in figure 3.3.

The results are not very surprising: computing the point errors requires computation of n^2 (pseudo-)random numbers, which is quite computationally intensive. Similarly, as a 1D FFT has a complexity of $\mathcal{O}(n \ln n)$, it follows that a 2D FFT has a complexity of $\mathcal{O}(n^2 \ln n)$, which is also fairly computationally intensive.

In practice the bottleneck in this investigation came from the time taken to plot the diffraction patterns - it could take quite a while to plot the results, especially with lots of padding where the central area of the pattern took up tens of thousands of pixels. Even in the most computationally intensive investigations in section 4.1, with a padding factor of 50, the time taken to compute that pattern was only around 5 seconds, much less than that to perform the analysis, adjust graphs and so on. It was decided that attempting to optimise further was not necessary.

3.4 Errors

The only sources of numerical error in the result should be an error due to discretisation (i.e. pixellation) of the aperture function, and an error due to discretisation of the diffraction pattern. It's not incredibly obvious how the pixellation of the aperture function should propagate through the computation, but it's clear that the pixellation error in the pattern is of the order of the angular extent of a single pixel in the pattern. Increasing the amount of padding increases the number of pixels in the diffraction pattern (since the FFT pattern has the same size as the input array), and so decreases the error in the pattern. Increasing the resolution, i.e. the number of pixels representing one square metre in the aperture function, will decrease the error resulting from pixellation of the aperture. As you can see in figure 5, the error due to pixellation of the result was typically very small.

4 Results

4.1 Correctness Checking

The first investigation performed was a thorough check that the computational framework produced results that were consistent with theory. First, the diffraction pattern was qualitatively analysed, by plotting the absolute value on a heat map. Logarithmic scaling was used for ease of visualisation. Since the diffraction pattern consisted of a very large array for any reasonable amount of padding, only the near-axis pattern was plotted. Initially a very large value was used for σ_T so that the well-know analytical results for a non-tapered circular aperture could be used for comparison. As can be seen in figure 4, qualitatively the pattern obtained is indeed an Airy disk. The simplest quantitative check on the correctness of the approach was to identify the angular extent at the first minimum in the pattern. Analytically[1], this should be equal to

$$\theta_{\min} \approx \frac{1.22\lambda}{D}, \tag{4}$$

where θ is in radians, and D is the diameter of the aperture. Presuming the algorithm behaves correctly, in order to check the algorithm was working as expected, the tapering was turned up to a factor of 100, and the resolution was put at 50 points per m^2. Measuring in degrees, the angular distance to the minimum was found to be

$$5.825 \pm 0.05 \times 10^{-3}, \tag{5}$$

taking the only error to be pixellation error in the pattern, compared to theoretical prediction of

$$5.825 \times 10^{-3}. \tag{6}$$

This is strong evidence that the implementation works as hoped.

4.2 Varying σ_T

In order to see how the diffraction pattern varied as σ_T varied, both the full width at half maximum extent (FWHM) and on-axis intensities were recorded for different values of σ_T. A qualitative look at what varying σ_T/R leads to can be seen in figure 4. Values for the tapering were chosen to be approximately linear in logarithmic space. Both exhibited linear behaviour when plotted on log-log axes, before saturating at high values of σ_T. For the FWHM measurements, it was possible to put error bars on the result; there was a clear measurement uncertainty due to the pixellation of the pattern. There was no clear measurement error for the intensity measurements as the intensity was just a number. The values of FWHM on varying σ_T can be seen in figure 5, plotted on log-log axes. We can clearly see a linear relation followed by saturation at high σ_T/R. This corresponds to the tapering becoming less and less relevant as the aperture approaches a uniformly transparent disk. The values of the on-axis intensity with different tapering can be seen in figure 6. Again we see a linear relation on the log-log plot, corresponding to a power-law. Indeed, the data fit a quartic curve very well.

4.3 Adding a hole

I then investigated adding a 0.5 m hole to the centre of the aperture. Using the routine explained above, I set the hole and plotted the resulting diffraction pattern. Figure 7 shows that the central

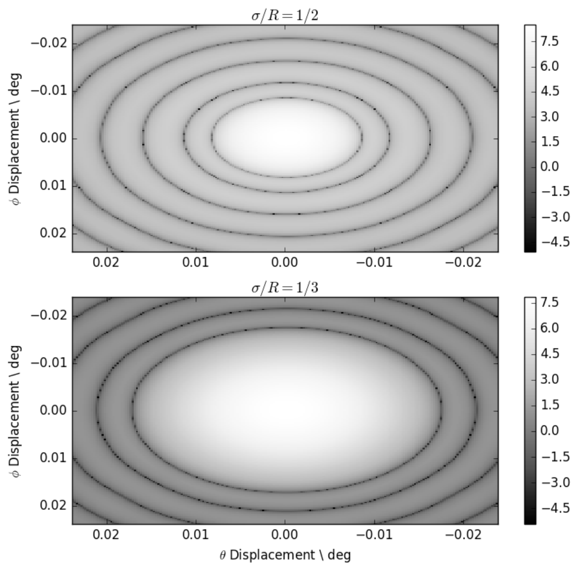

Figure 4: A plot showing the qualitative dependence of the diffraction pattern on σ_T. The log intensity is plotted in arbitrary units. A smaller σ clearly causes the central maximum to become enlarged. The on-axis intensity also drops, although this is not as clear from this plot.

maximum was not changed appreciably (and in fact the on-axis intensity only changed by a factor of 4%). However the first minimum was substantially enlarged, spanning a ring with much more area compared to the minima of the aperture with no hole.

4.4 Adding Point Errors

Here, a phase error was added to every point in the aperture. The errors tended to distort the diffraction pattern, as can be seen in figure 8. Quantitatively looking at the on-axis intensity as a function of the standard deviation of the distribution of the errors, σ_ε, we see an interesting pattern, shown in figure 9. At low error, the central maximum power is constant, but drops sharply between a $\sigma_\varepsilon/\lambda$ ratio of 0.05 and 0.2, before fluctuating at a much lower power for larger values. This corresponds to a phase error of 0.2 to 0.8 π radians. It is clear that this marks the transition from a phase distributed tightly around 0, to a phase error that leads to an essentially uniformly randomly distributed phase at each point. It is no surprise that this high amount of phase variation leads to a much reduced power on-axis.

4.5 Adding Extended Errors

Adding errors at a point is well and good, but a more reasonable model of physical errors in a radio telescope could take the form of randomly-distributed extended errors with a Gaussian profile. The beam pattern was analysed as a function of the correlation length of these errors. However, a longer correlation length would naturally result in a greater phase error integrated under the gaussian, and so some care has to be made to separate the effect of the distribution from the effect due to the total amount of phase error. The volume under the gaussian is proportional to l_c^2, so the number (n) of such gaussian errors imposed on the aperture was scaled so that nl_c^2 was kept constant. It was very hard to produce quantitative results for this part, but qualitatively we can say that the errors with a large extended length scale had a more distruptive effect on the resulting diffraction pattern than errors with a smaller extended length scale. This can be seen in figure 10. This is perhaps somewhat expected: an error consisting of a large blob can have a coherent effect on the diffraction pattern, while a very small l_c essentially reduces to the case of the point errors in section 4.4, where the errors contribute incoherently to any error in the pattern.

5 Conclusion

It has been shown that it is possible to generate the Point Spread Functions of various apertures numerically using a Fast Fourier Transform in the Python language. Furthermore, this method easily extends to complicated apertures functions that model non-ideal lenses. The effect of a 'tapered' aperture function where $A(r) \propto e^{-r^2/\sigma_T^2}$was modelled - it was found that the on-axis intensity fell as $I \propto (\sigma_T/R)^4$, whilst the Full Width at Half Maximum amplitude (FWHM) went as $W \propto (\sigma_T/R)^{-1}$, although both effects levelled off at high σ_T. Qualitative analysis of the effect of a central hole in the mirror was shown in figure 7, with enlargement of the first minimum and some loss of definition for further minima. Analysis of independent Gaussian-distributed errors at each point in the aperture was carried out and displayed in figure 4.4: it was found that the on-axis intensity was steady for $\sigma_\varepsilon/\lambda$ less than 0.1, at which point it dropped by a factor of 10^4. Finally, extended errors with a Gaussian profile were added to the aperture and qualitative results were shown in figure 10.

Further analysis could apply Fourier methods to compute the patterns of extended sources, using the wealth of literature available on the topic [7], or use the computed PSF to correct for distorted mirrors.

6 Plots

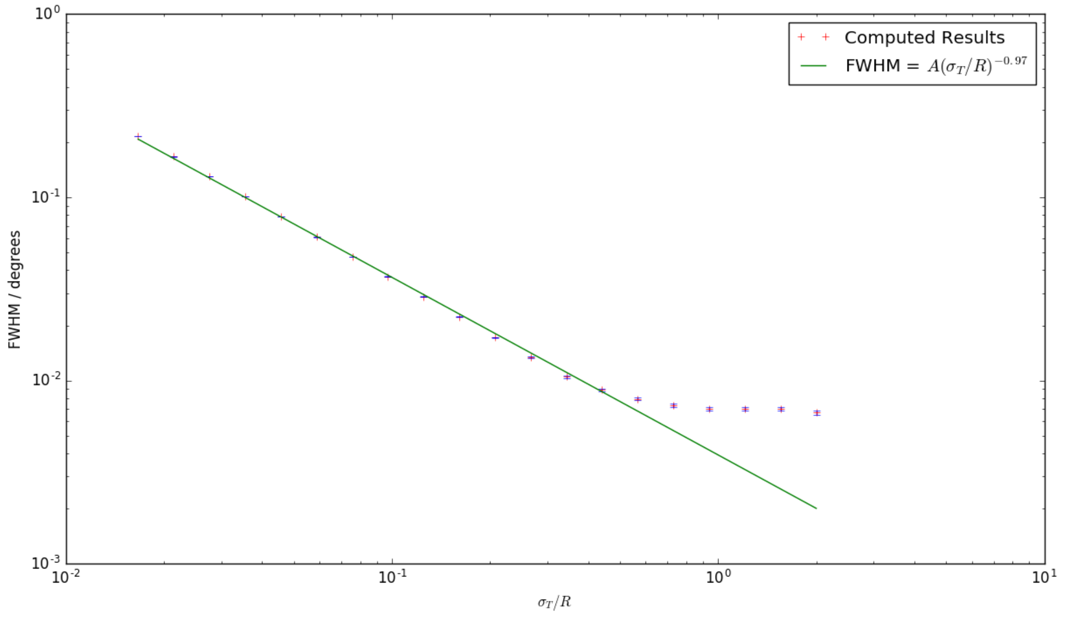

Figure 5: A plot showing the dependence of the full-width at half maximum amplitude to the ratio σ_T/R. As can be clearly seen, for a large range of σ_T/R the computed result closely follows $FWHM \propto (\sigma_T/R)^{-1}$ before levelling off. The angular extent of one pixel was used for the error bars.

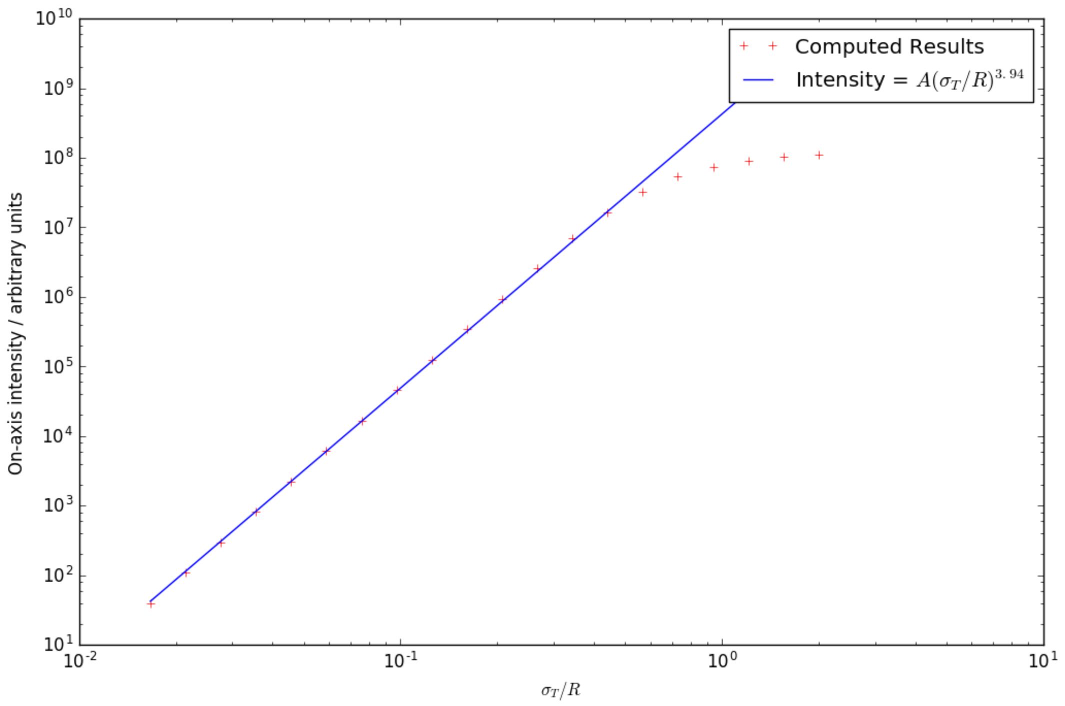

Figure 6: A plot showing the dependence of the on-axis amplitude (in arbitrary units) to the ratio σ_T/R. As can be clearly seen, for a large range of σ_T/R the computed result closely follows $I \propto (\sigma_T/R)^4$ before levelling off.

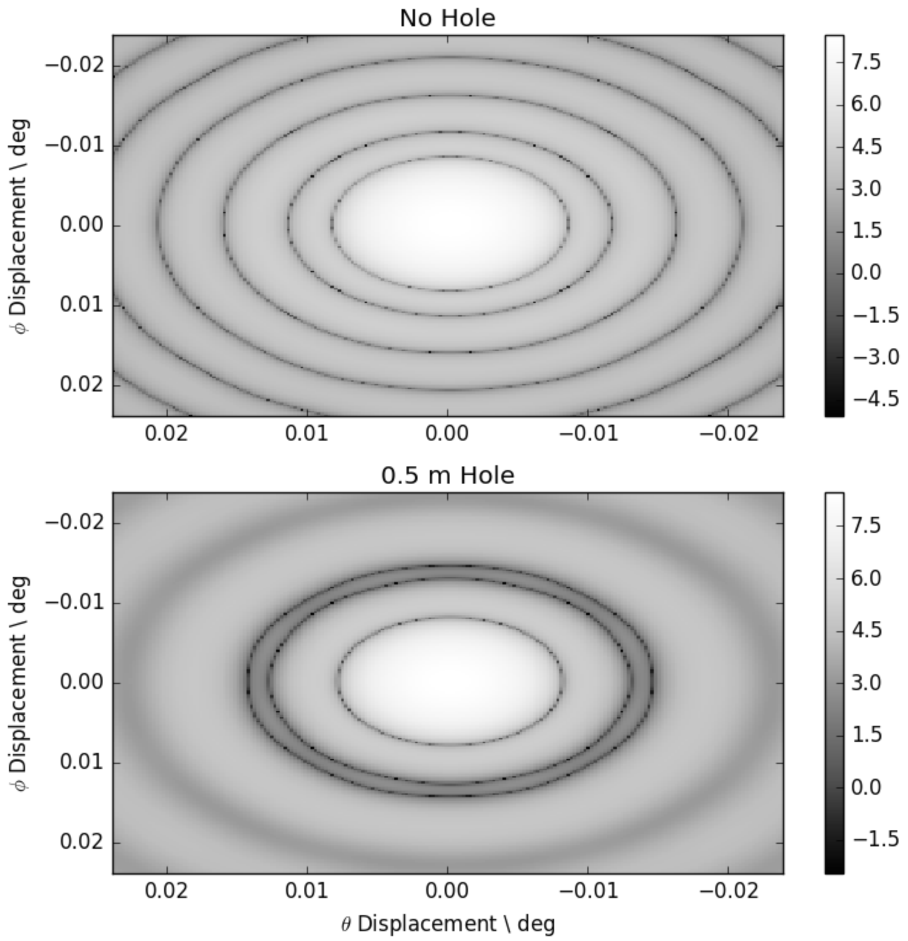

Figure 7: A plot showing the result of applying a 0.5m hole to the aperture function. The first minimum is much enlarged and the subsequent minima not as clearly defined.

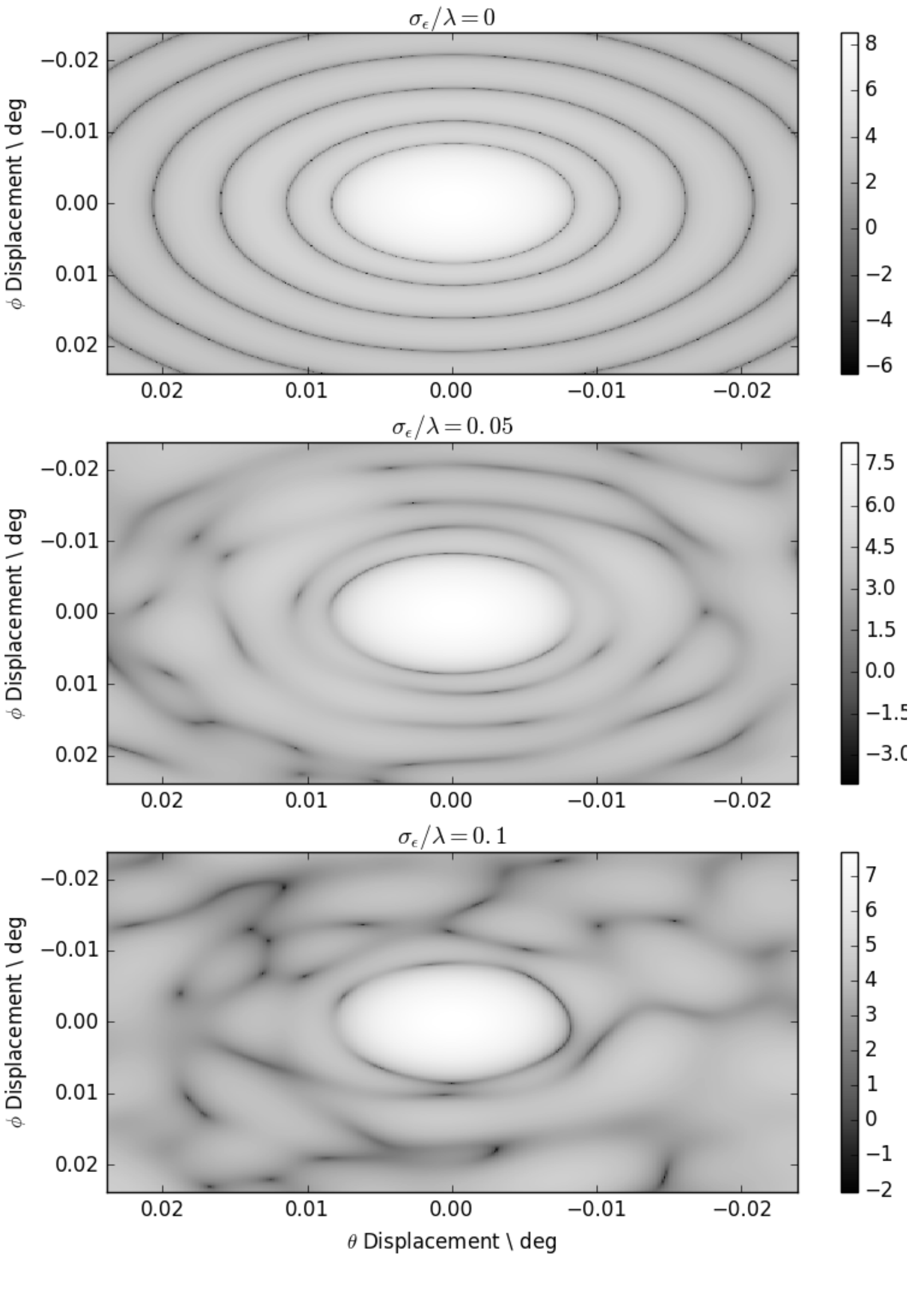

Figure 8: A plot showing the qualitative dependence of the diffraction pattern on σ_ε. As it increases, the secondary minima become less defined, and the central maximum starts losing definition too.

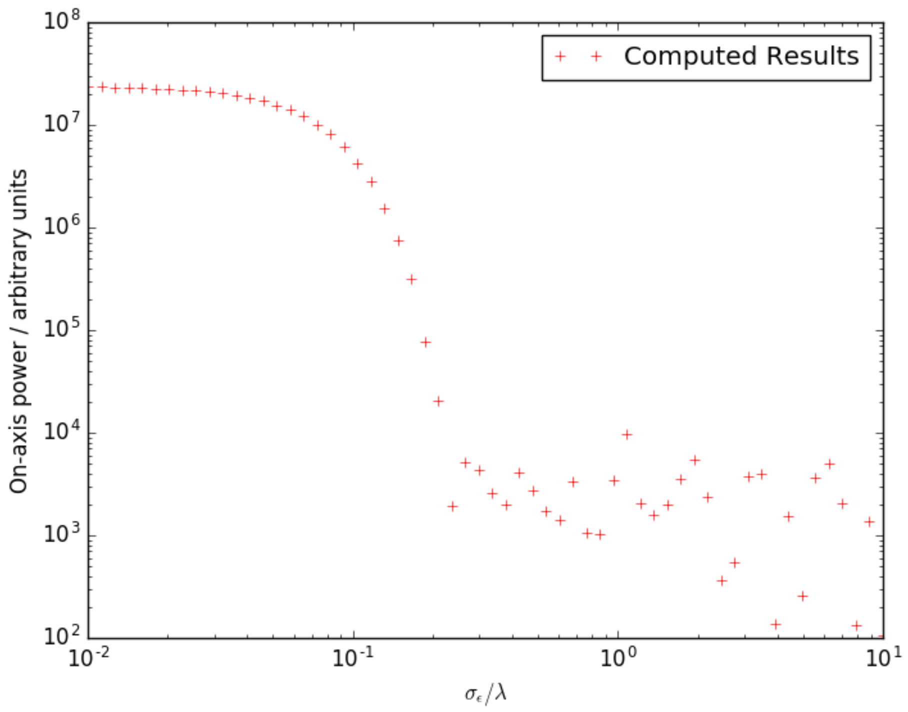

Figure 9: A plot showing the dependence of the central power on $\sigma_\varepsilon/\lambda$. At low error, the central maximum power is constant, but drops sharply between $\sigma_\varepsilon/\lambda$ of 0.1 and 0.2, before fluctuating at a much lower power for larger values.

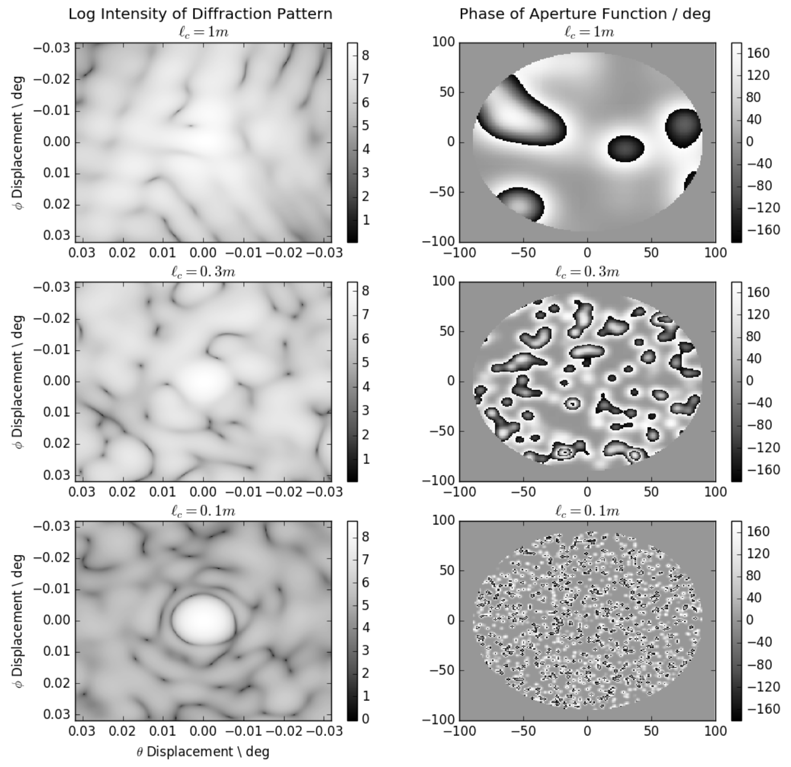

Figure 10: In this figure we see the effect of adding extended phase errors to the aperture, each with a Gaussian profile and coherence length l_c. The number of errors was scaled so that the total phase error was the same in all cases. We can see that the longer l_c values lead to more extreme distortions; particularly interesting is to look at the increased definition of the central maximum as l_c gets shorter and shorter.

A Code Listing

The functions used to generate the diffraction patterns and add errors etc are reproduced here in full. For brevity, full code to produce every plot shown here is not listed. Example code to produce figure 4 above is shown.

```
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib


class Mirror:
    """Class encapsulating a mirror. The class contains a complex
    array representing the aperture function and another (complex)
    array may be generated representing the far-field diffraction
    pattern"""

    points_per_m = 10
    pad_factor = 20
    #Number of times larger the padded array sides are compared to 2R
    #e.g. pad_factor = 1 corresponds to no padding.


    def taper(self, x, y, sigma, side):
        """Given an x,y cell index for an array with side length side, return
        the value of a 2-D Gaussian function centred on the centre of the
        array with standard deviation sigma at the point x,y. All values
        are in terms of cell indices."""
        return np.exp(-((x - side / 2.0) ** 2 +
                        (y - side / 2.0) ** 2)
                      / (2 * sigma ** 2))

    def __init__(self, R, sigma_T):
        """Initialise the class, generating the aperture function array
        as a circle of radius R, tapered with the supplied value of sigma.
        """
        self.sigma_T = sigma_T
        self.R = R
        self.L = R * self.points_per_m #Number of array cells in a radius
        self.side = 2 * self.pad_factor * self.L
        #Number of cells in the padded aperture array
        self.aperture = np.fromfunction(lambda i, j:
                                        self.taper(i, j,
                                                   self.sigma_T *
                                                   self.points_per_m,
                                                   2 * self.L),
                                       (2 * self.L,
                                        2 * self.L), dtype=complex)
        #Above line generates a square array where each cell has the
        #value obtained by calling taper on that x,y coordinate

        y,x = np.ogrid[-self.L : self.L,
                       -self.L : self.L]
        mask = x*x + y*y >= self.L ** 2
        self.aperture[mask] = 0
        #Set cells at radius > R to 0

        #Embed in padded array
        padded = np.zeros([self.side, self.side], dtype=complex)
        padded[self.side / 2 - self.L: self.side / 2 + self.L,
               self.side / 2 - self.L: self.side / 2 + self.L] = \
                                                  self.aperture
        self.aperture = padded

    def pattern(self, wavelength):
        """Calculate the far-field diffraction pattern with an FFT"""
        self.pattern = np.fft.fft2(self.aperture)
        self.pattern = np.fft.fftshift(self.pattern)
        self.scaling_factor = float(wavelength) / (float(self.side) /
                                           self.points_per_m)
        #Angular width that a pixel corresponds to in the pattern array
        #in radians

    def find_width(self, power):
        """Given a power, finds the width between the two points in the
        central row of the pattern where the points have the value closest
        to power. NB assumes symmetric pattern, returns width in degrees."""

        #First fix up any problems that could occur due to phase differences
        row = np.absolute(self.pattern[len(self.pattern) / 2])
        power = abs(power)
        index = np.argmin(np.absolute(row - power)) #find index of value
        return abs(index - len(self.pattern) / 2) * 2 * (self.scaling_factor
                                                 * 180 / math.pi)
        #Return physical angular width in degrees


    def add_hole(self, radius):
        """Adds a hole of radius r to the telescope, by setting the
        aperture function to zero in this region."""
        y,x = np.ogrid[-self.side / 2:self.side / 2,
                       -self.side / 2:self.side / 2]
        mask = x*x + y*y <= (radius * self.points_per_m) ** 2
        self.aperture[mask] = 0

    def add_point_errors(self, sigma_eps, wavelength):
        """Multiplies every point in the aperture by a phase error
        4*pi*epsilon/wavelength, where each epsilon is gaussian-disistributed
        with standard deviation sigma_eps"""
        epsilon_array = np.random.normal(0, sigma_eps, [(self.side),
                                                        (self.side)])
        phase_array = np.exp(1j * epsilon_array * 4 * np.pi / wavelength)
        self.aperture = self.aperture * phase_array


    def add_large_deformations(self, l_c, max_amp, n, wavelength):
        """Adds n 2-D gaussian functions with characteristic length l_c
        (= sigma) and uniformly distributed amplitude to the aperture"""
        #Cut the Gaussians off at 3 sigma
        for i in range(0,n):
            #Select random coordinates in the grid. We choose ones that are
            #in the rectangle that the aperture is inscribed in
            array_l_c = int(math.floor(self.points_per_m * l_c))
            x, y = np.floor(np.random.random(2) * 2 * self.L +
                            (self.pad_factor - 1) * self.L)
            amplitude = np.random.random() * max_amp
            gauss_array = amplitude * np.fromfunction(lambda i, j:
                                                      self.taper(i, j,
                                                                 (array_l_c),
                                                                 6 * array_l_c),
                                                      (math.floor(6 * array_l_c),
                                                       math.floor(6 * array_l_c)),
                                                      dtype=complex)

            self.aperture[x - 3 * array_l_c: x + 3 * array_l_c,
                          y - 3 * array_l_c: y + 3 * array_l_c] = \
            (np.exp(4 * 1j * math.pi * gauss_array / wavelength) *
             self.aperture[x - 3 * array_l_c: x + 3 * array_l_c,
                           y - 3 * array_l_c: y + 3 * array_l_c])
```

```
#--------------------------------------------------------------------------------
#Example of plotting code - reproducing figure 4 in the report
plt.subplot(2,1,1)
taper = Mirror(6,3)
taper.pattern(0.001)
mid = len(taper.pattern) / 2
x = np.linspace( -100 * taper.scaling_factor * 180 / np.pi,
                  100 * taper.scaling_factor * 180 / np.pi,
                 200)
#x-axis in degrees

#Fixes plot showing odd whitespace
plt.xlim([x[0], x[-1]])
plt.ylim([x[0], x[-1]])

plt.title(r'$\sigma/R = 1/2$')
plt.ylabel(r'$\phi$ displacement \ deg')

#Take central 200 * 200 pixels
plt.pcolor(x, x, np.log(np.absolute(taper.pattern[mid - 100: mid + 100,
                      mid - 100: mid + 100])),
              cmap='Greys_r')
plt.colorbar()

plt.subplot(2,1,2)
taper = Mirror(6, 2) #Re-generate with sharper tapering
taper.pattern(0.001)
plt.pcolor(x, x, np.log(np.absolute(taper.pattern[mid - 100: mid + 100,
                      mid - 100: mid + 100])),
              cmap='Greys_r')
plt.colorbar()
plt.ylabel(r'$\phi$ displacement \ deg')
plt.xlabel(r'$\theta$ displacement \ deg')
plt.title(r'$\sigma/R = 1/3$')

#Fixes plot showing odd whitespace
plt.xlim([x[0], x[-1]])
plt.ylim([x[0], x[-1]])


plt.show()
```

References

[1] *On the Diffraction of an Object-glass with Circular Aperture* Airy, G. B., Transactions of the Cambridge Philosophical Society, Vol. 5, 1835, p. 283-291.

[2] *Fast Fourier Transform* Wolfram MathWorld, `http://mathworld.wolfram.com/FastFourierTransform.html`, Retrieved 2016-03-23.

[3] *Astronomical Optics* Daniel J. Schroeder, Academic Press, Inc. 1987.

[4] *The Hubble Telescope - Optical Systems Failure Report* L. Allen et al. NASA/JPL, November 1990

[5] *Part II Computational Physics Exercises* David Buscher, Course Handout, February 2016. Physics Department, Cambridge University.

[6] *Python Call Graph* `https://pycallgraph.readthedocs.org/en/master/` Retrieved 2016-03-23.

[7] *Diffraction-Limited Imaging with Very Large Telescopes* D. M. Alloin and J.-M. Mariotti Kluwer Academic Publishers 1988.