# Optics with Imperfect Mirrors
# Part II Computational Project

6th April 2019

**Abstract**

Write fancy abstract here TODO.

---

## 1 Introduction

Modern telescopes use very finely polished curved mirrors to create sharp images. However, errors are an inherent property of any physical system and any manufacturing process. Thus, regardless of the way mirrors are manufactured (e.g. spin-casting or polishing), there are still imperfections in their shape.

The aim of this paper is a computational investigation of the effects of imperfections in the shape of a telescope mirror on the image. Several relationships are studied, as suggested in the projects manual [1]:

- the effects of Gaussian illumination, where the amplitude of radiation on the mirror

$$|A(r)| \propto \exp\left(-r^2/2\sigma_T^2\right)$$

  on the size and central intensity of the image.
- the effect of a central hole in the mirror, due to the secondary mirror.
- the effects of random and correlated phase errors, due to physical dents in the surface of the mirror. The dents cause phase errors due to the change in path length with respect to the smooth shape.

A more detailed analysis of the theory and techniques used is given in Section 2. Implementation details are discussed in Section 3, and results are presented and discussed in Section 4.

## 2 Analysis

In the analysis of this problem, we treat the far-field diffraction pattern of the mirror under uniform or Gaussian illumination. In the far field regime, the diffraction pattern of an aperture is given by the Fourier Transform of the aperture function $A(x, y)$ [3, Chapter 10.2]. This is a reasonable approximation as long as:

$$\frac{R^2}{\lambda D} \ll 1, \tag{1}$$

known as the Fraunhofer limit, where $R$ is the maximum extent of the aperture, $\lambda$ is the wavelength of radiation and $D$ is the distance from the aperture to the screen onto which the image is projected. In this problem we use:

$$\lambda = 1\,\text{mm}$$
$$R = 6\,\text{m},$$

which would require a distance on the order of 1 km for the far-field limit to hold.

However, we can work in angular coordinates, with the image space spanned by:

$$p = k \sin\theta \approx \frac{kx'}{D}$$
$$q = k \sin\chi \approx \frac{ky'}{D},$$

where $x'$ and $y'$ are distances that would span a physical image space, and $k$ is the wavenumber. We thus eliminate the dependence on $D$. In these coordinates, the image is given by:

$$\psi(p,q) \propto \iint A(x,y) \exp\left(ipx + iqy\right) dxdy, \tag{2}$$

which is just a Fourier transform of $A$. Seemingly we've also eliminated the dependence on $\lambda$, but that will be needed again in the analysis of dented mirrors (Section 2.3).

## 2.1 Discrete Fourier Transforms

The fact that the image is a Fourier transform of the aperture function is very useful. The Fast Fourier Transform algorithm can compute discrete Fourier transforms of multi-dimensional data efficiently, and there exist library implementations of it for virtually any language. Here, the C++ `FFTW 3` library [2] was chosen, as suggested in the projects manual [1]. It is a well-established and well-tested library with very good computational efficiency.

Minor adaptations are required to use the DFT algorithms in the library. The 1D discrete Fourier transform is defined as [4, Chapter 12.1]:

$$H_k \propto \sum_{n=0}^{N-1} h_n \ \exp\left(2\pi i \frac{kn}{N}\right), \tag{3}$$

with the corresponding frequency values:

$$f_k = \frac{k}{N\Delta}, \quad k = 0..N-1, \tag{4}$$

where $\Delta$ is the sampling interval of the original signal. Comparing to a 1D discrete form of the diffraction integral (Equation 2):

$$\psi_k \propto \sum_{n=0}^{N-1} A_n \exp\left(ip_k x_n\right), \quad \text{where} \quad x_n = n\Delta \tag{5}$$

we see that we need to rescale:

$$p_k = 2\pi f_k \tag{6}$$

## 2.2 Testing

To determine whether the algorithm is outputting something sensible, we need to test it on a range of known results. Diffraction patterns for several kinds of apertures are easy to compute analytically, and thus can be used for testing the program.

**Rectangular aperture** A rectangular aperture of size $-a < x < a$, $-b < y < b$ has diffraction pattern:

$$\psi(p,q) \propto \frac{\sin\left(pa\right)}{pa} \frac{\sin\left(qb\right)}{qb}, \tag{7}$$

thus it's expected to have the first zeros at:

$$p = \pm\frac{\pi}{a} \quad \text{and} \quad q = \pm\frac{\pi}{b}. \tag{8}$$

**Circular aperture** A circular aperture of radius $R$ has a diffraction pattern called an Airy disc, with angular radius:

$$\sin\theta \approx 1.22\frac{\lambda}{2R} \quad \Rightarrow \quad p = 1.22\frac{\pi}{R}. \tag{9}$$

Both of these zero-intensity points should appear in the diffraction patterns, and the scaling with the inverse of the aperture size should be observable.

### 2.3 Bent mirrors

## 3 Implementation

The implementation consists of several different parts, discussed in the subsections that follow. The overall logic flow of the program is: The executable is invoked with one argument, pointing to a configuration file. The instructions therein are read and executed in order, thus computing diffraction patterns for the described apertures. The results are printed to disk, and figures can then be produced by invoking the respective Python scripts for each problem.

C++ source files are in `src/cpp`, and plotting scripts in `src/scripts`. Data is written by convention to the `data` directory, and figures to `fig`.

### 3.1 Describing Apertures and Images

In the `FFTW` library two-dimensional $N_x$ by $N_y$ arrays are represented as one-dimensional $N_x \times N_y$ arrays of complex numbers (of the inbuilt `complex` type) [2, Section 3.2].

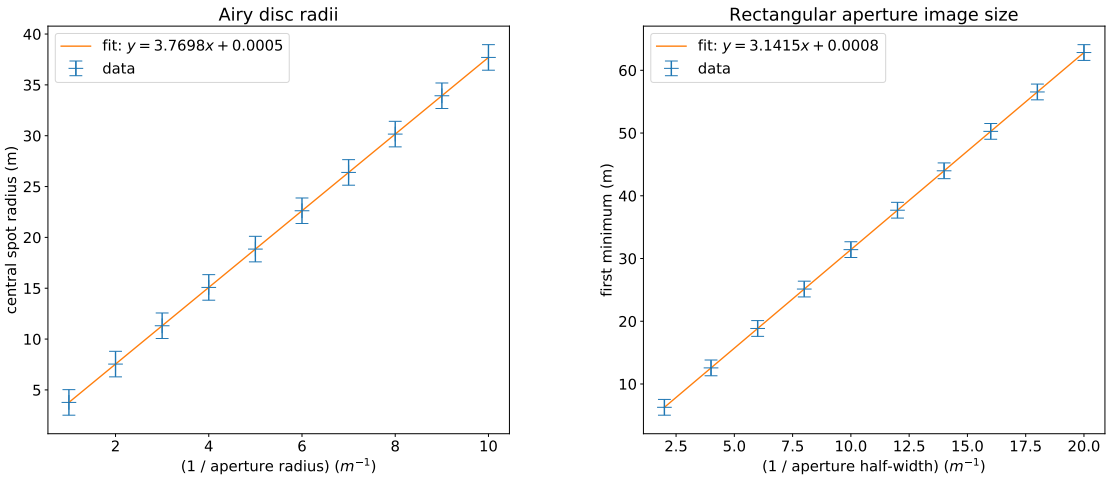
Because working directly with a 1D representation of 2D data can be clunky, I decided to wrap this functionality in a class called `Array2d`, declared in `array2d.h` and implemented in `array2d.cpp`. The class stores the data internally in the 1D array representation, but has a more user-friendly interface. For example, it defines the `[i][j]` and `(i, j)` operators for easy access to the element in the $i^{th}$ row and $j^{th}$ column.

**FFTW compatibility** The method `ptr()` returns a pointer to the beginning of the array, cast to `fftw_complex` type. This pointer can be passed to functions in the `FFTW` library [2, Section 4].

**Memory allocation** One special feature of this class that breaks with convention is that there is no explicitly defined copy constructor. The compiler generated one only performs a shallow copy, copying the pointer to the data array, but not the data itself. Only the explicitly defined constructor `Array2d::Array2d(int nx, int ny)` uses `fftw_alloc_complex` to allocate new memory for the array. This means that the user of the class has finer control over where memory is allocated, which is important in the case of large arrays. For example, using IEEE double-precision floating point, which occupies 64 bits of memory, a $2^{13} \times 2^{13}$ array of complex numbers occupies around 1 GB of memory.

For better understanding of when new memory is allocated, compile the code with the variable `DEBUG_OUT` set to `true` in `array2d.cpp`. Then the calls to the memory-allocating constructor of `Array2d` will be logged to standard output.

**Aperture generators** These are a type of function that initialise an `Array2d` with a particular kind of aperture with given parameters. For example, there is a generator for circular apertures, one for circular Gaussian-illuminated, one for circular with random errors, etc.




(a) (b)

Figure 1: Sizes of central spots as function of aperture size, for both round and rectangular apertures.

### 3.2 Configuration Files

Configuration files are stored by convention in the `config` directory, and are a series of `key = value` lines. They contain:

- `nx` and `ny`. These are the size of the problem, i.e. the numbers of rows and columns that the mirror shape is split into.
- `tasks`, what actions to perform on each of the shapes
- `n_shapes`, the number of shapes
- for each shape, it's properties: `type`, `lx` and `ly`, the limits within which $A(x, y)$ is defined, and `params`, a list of parameters of the shape, e.g. the dimensions or the taper of Gaussian illumination.

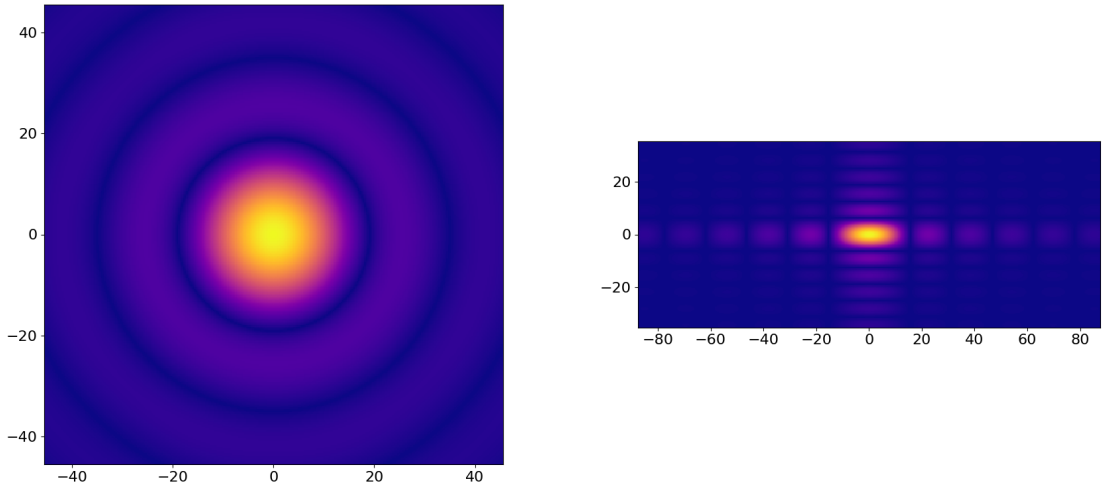
### 3.3 The `fftshift` operation

The result of this operation is shifting the zero-frequency component of the spectrum from the beginning to the middle of an array. This takes advantage of the fact that the Fourier frequencies (Equation 4) are cyclic with period $1/\Delta$ [4, Section 12.1.2]. The function `fftshift` is overloaded to act on both 1D `vector` and 2D `Array2d` objects.

## 4 Results and Discussion

### 4.1 Tests

First, I ran the program on series of circular and rectangular apertures described in `config/circular_mins.txt` and `config/rectangle_mins.txt`, expecting to see a linear relationship between the extent of the central maximum and the inverse of the aperture size (see Section 2.2). As seen in Figure 1, these linear relationships hold. The line slopes are close to the expected values of $1.22\pi$ for the Airy disc and $\pi$ for rectangular shapes.




(a) (b)

Figure 2: Typical diffraction patterns of circular and vertical rectangular apertures.

Figure 2 shows that the diffraction patterns look as expected. Most notably, the image from a vertical rectangle is longer along the horizontal axis, as expected from Equation 8.

# 5 Conclusion

# References

[1] David Buscher. *Part II Computational Physics Projects*. Cavendish Laboratory, University of Cambridge. Lent 2019. URL: `https://www-teach.phy.cam.ac.uk/dms/dms_getFile.php?node=20927` (visited on 03/04/2019).

[2] Matteo Frigo and Steven G. Johnson. *FFTW. for version 3.3.8*. Massachusetts Institute of Technology. May 2018. URL: `http://www.fftw.org/fftw3.pdf` (visited on 04/04/2019).

[3] Eugene Hecht. *Optics*. 5th edition. Pearson Education, 2017. ISBN: 978-1-292-09693-3.

[4] William H. Press et al. *Numerical Recipes. The Art of Scientific Computing*. 3rd edition. Cambridge University Press, 2007. ISBN: 9780521880688.