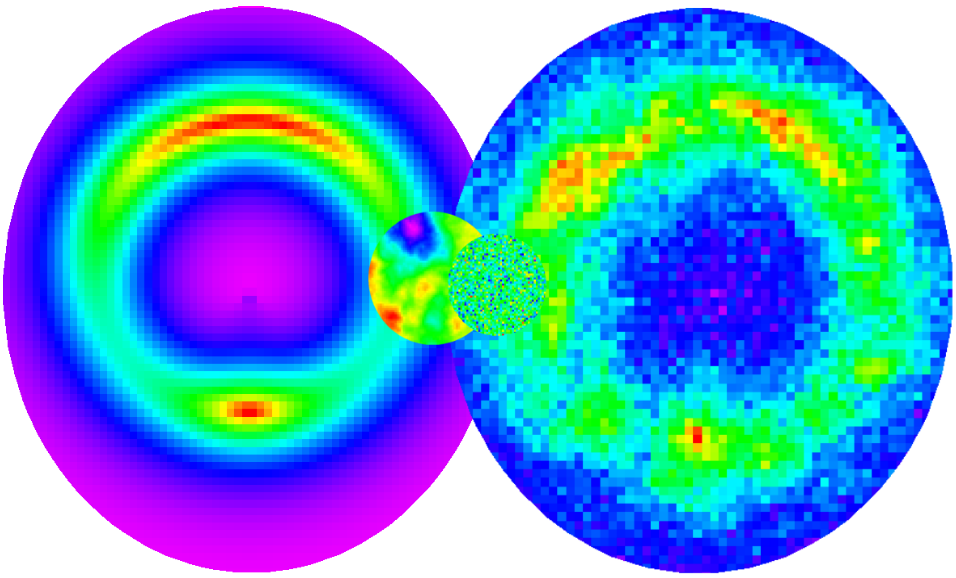
university of groningen / faculty of mathematics and natural sciences / kapteyn astronomical institute

Bachelor Thesis



Numerical Simulations of Lens Anomalies

Author:
Robin R. Kooistra

Supervisor:
Prof. dr. L.V.E. Koopmans

July 18, 2012

Abstract

The concept of gravitational lensing describes the deviation of a light ray from a straight line due to the presence of a gravitational field. This results in distorted and magnified images of distant object. We perform simulations of gravitational lenses using where we perturb the lens potential using Gaussian random fields. By subtracting a non perturbed model from the simulations intensity fluctuations are created that are caused by the potential fluctuations. Using a power spectrum analysis we determine under which conditions it will be possible to measure the effect of the potential fluctuations from these residuals. The simulations were performed for several noise levels, different sizes of the source object and multiple scales of the fluctuations. Finally we also vary the slope of the power spectrum of the potential fluctuations. Our results show that a high S/N ratio is required to be able to extract the power spectrum due to the fluctuations from the noise. Smaller scale fluctuations cause the power spectrum to be more easily dominated by noise and increasing the source size has a small effect on the amplitude of the measured power spectrum, but not on the overal shape. The only two situations that produce measurable spectra with a confidence higher than 90% are for the two largest sources ($\sigma_{src} = 0.5$ and 0.8) with a noise level of $\sigma_{noise} = 5.0$ intensity units (corresponding to a mean S/N ratio over the image of $\sim$ 7-9). Increasing the steepness of the potential fluctation power spectrum result in a cut-off at smaller scales, making noise again dominant there. Multiple observations of lens events will be required to get a statistically accurate measument of the entire power spectrum. For single observations only the large scale side of the power spectrum will be measurable. Future work will be necessary to compare the results with an analytical solution and to determine the optimum observation strategy for HST, Keck Adaptive Optics, EUCLID or other telescopes.

Contents

Chapter 1

Introduction

The theory of General Relativity has been a stimulation for various areas of science. Its number of applications in astronomy alone are vast to say the least. One of the results from Albert Einstein's theory is that the path that light travels can be altered due to gravity. This effect can cause images of distant objects to be distorted by the presence of a large mass along the line of sight. The theory describing the distortion of these sources is called 'Gravitational Lensing' and can be useful to determine properties of the Universe and both the objects at high redshifts and the massive ones (called 'lenses') that cause this bending of light[1]. It is the latter that will be studied and discussed in the research described in this thesis.

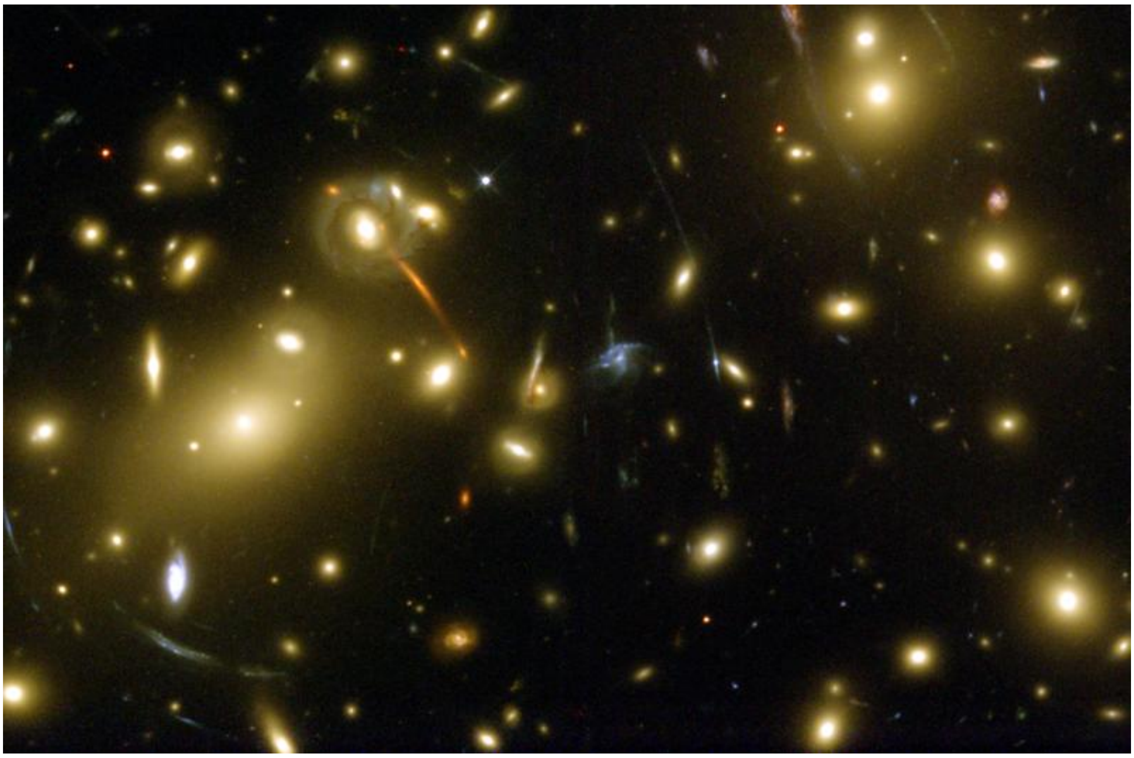


Figure 1.1 – The Abell 2218 cluster of galaxies. Lensing events are evident here with ring or arc-like structures clearly visible.[2]

1.1 Gravitational Lensing In A Nutshell

It has often been wrongly stated that Newton already thought about the bending of light by gravity. Literature studies have shown that in fact he was referring to diffraction[3]. Nonetheless, ideas on gravitational lensing began early in history. The first explicit mention of the bending of light due to gravition was made by Henry Cavendish at the beginning of the 19th century[3]. Around the same time, Soldner also performed calculations of the deflection of light by the Sun[4]. After that, the idea of gravitational lensing was not described much further, until 1911. That year, Einstein published a detailed prediction of the bending of light, where he used the equivalence principle to derive the result from Cavendish and Soldner to first-order [3]. This result was confirmed in 1919 with measurements of the shift in the position of stars around the edge of the Sun[4]. Analysis of notebooks of Einstein at the same time show his derivation of the lensing equation[3]. Furthermore they also contain sketches of the position of gravitationally lensed images. Later work by Eddington used an analogy between gravitational deflection and refraction to derive earlier predictions in simple terms and looked at the possibility of multiple images of a source[1][3]. His work inspired Link to do some detailed computations, who was confident in the possibility of observing the effect[3]. In 1936 (nine months after Link's work was published), Einstein published his famous paper in which he concluded that the lensing effect of stars by other stars would not be observable, because the angular separation would be too small[3][1]. The next year, Zwicky considered the possibility of galaxies to act as lenses. His calculations showed that the chances of observing a lens event were actually not as low as predicted earlier[4]. It still took more than 40 years for the first true observation of a lensed object to appear (see figure 1.2)[1].

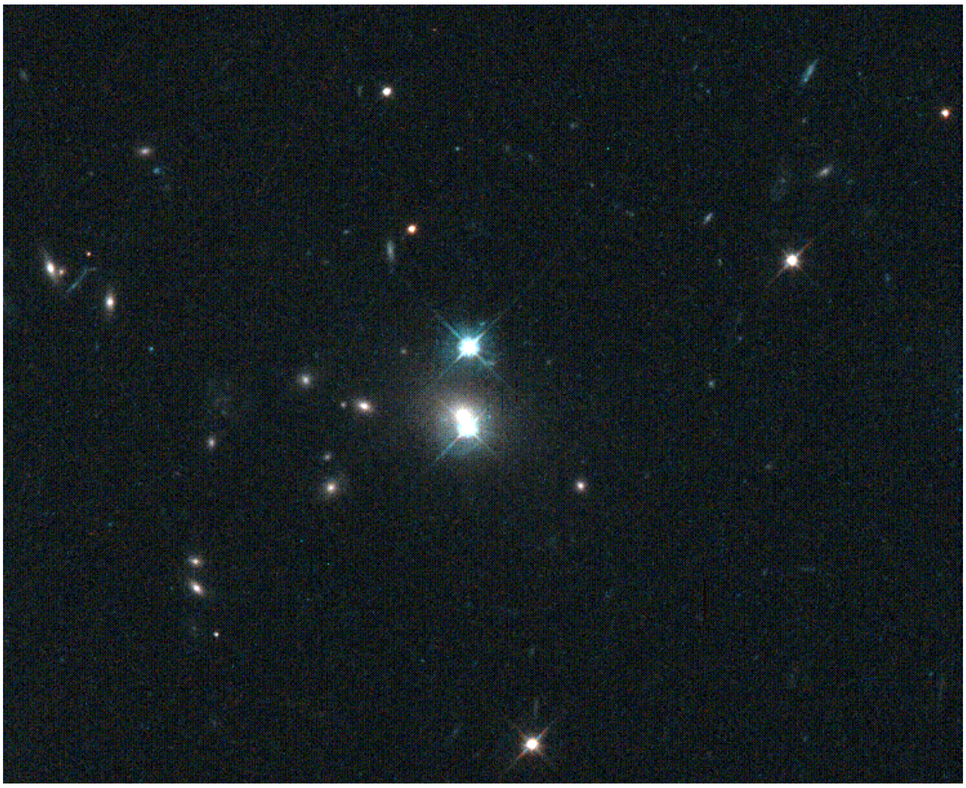


Figure 1.2 – Image of QSO 0957+561. The first ever oberved lensed quasar, by Walsh, Carswell & Weymann (1979). The two images have an angular separation of ~ 6" and their redshift was measured at z = 1.41 [1][5]

In the years that followed, many more lenses were discovered. A lot of them are multiply imaged systems and some show an arc or ring-like structure. These are strongly distorted images shaped like a ring or arc around the lens, visible in for example the Abell 2218 cluster seen in figure 1.1[1]. Currently, hundreds of lensed objects have been discovered. An incomplete database of Hubble Space Telescope and radio images of lenses from the CASTLES survey can be found online [6]. It shows that, there is a large variety of different shapes of images that arise through gravitational lensing.

Besides giving these kinds of images, the lens events also have some useful applications. They can help constrain cosmological parameters like the Hubble constant[1]. Furthermore, the deflection angle due to lensing only depends on the mass distribution of a lens and not on its luminosity. Therefore lensing by dark matter makes it a useful way to detect this unknown substance[4]. Last, but not least, the lenses can act as a magnifying glass. Especially galaxy clusters are able to reveal very distant objects that would normally not be visible with present-day telescopes[4]. Some of the farthest away galaxies have been observed through gravitational lensing. An example is a redshift 9.6 galaxy in the MACS1149+22 cluster discovered in 2012 [7]. For a more complete summary of the applications of gravitational lensing, the reader is refered to the article by Schneider (2003)[1], the article by Treu (2010)[8] and the lectures by Narayan (2008)[4].

1.2 The Goal

The focus of this thesis is on simulating and quantifying the effects of small-scale density fluctuations in lens galaxies using numerical simulations. We perform simulations of lensing events and add perturbations to the lens with fluctuations of different scales representing different kinds of small-scale structures. From this we will try to see if it will be possible to accurately measure the effect of the fluctuations.
The project was performed as a final assignment for the Bachelor in astronomy. We begin with derivations of the important concepts that are required for the simulations, including the mathematical tools involved. Chapter 3 then goes into the numerical implementation of the model, the results of which are discussed afterwards. We finish with a conclusion and explain some of the work that can be done to follow up on this research.

Chapter 2

The Concepts Involved

To understand what happens in the simulations, some basic concepts first need to be introduced. In the next couple of sections we derive some important relations and give a description of the mathematical tools that were used. This is done in preparation of the numerical methods introduced in the next chapter.

2.1 The Lens Equation

Gravitational lensing describes how a light ray deviates from a straight line due to the curvature of space time around a massive object. From General Relativity it follows that the angle by which a light ray is reflected by the gravitational potential of a point mass is given by the following relation[4].

$$\hat{\alpha} = \frac{4GM}{c^2\xi} \tag{2.1}$$

Here ξ is the closest distance the light ray passes to the object. In general the objects that cause the lensing are not point masses, but instead are extended and have a certain mass distribution. To get the deflection angle for such an object we can superpose the angle arising from the individual mass elements of the lens.

$$\hat{\vec{\alpha}}(\vec{\xi}) = \frac{4G}{c^2}\int d\vec{\xi}'\Sigma(\vec{\xi}')\frac{\vec{\xi}-\vec{\xi}'}{\left|\vec{\xi}-\vec{\xi}'\right|^2} \tag{2.2}$$

Now we integrate over the surface density $\Sigma(\vec{\xi}')$ (the integral of the mass density over the line of sight) and $\left|\vec{\xi}-\vec{\xi}'\right|$ gives the impact parameter for interaction with a mass element. This is only valid under the assumption that the deflection angle is small which requires the lens to be much smaller than the distances between the source, lens and the observer[1]. The assumption described here is known as the geometrically thin lens approximation and is in general satisfied for the objects considered like (clusters of) galaxies.

Using the deflection angle it is possible to derive the lens equation, which relates the position where the image of a source forms to its actual position. Figure 2.1 gives a schematic representation of what a basic lens system looks like. Normally light from the source at position η in the source plane would travel in a straight line, where it would reach the observer under an angle $\vec{\beta}$. Because of the presence of the lens, it is now deflected by an angle $\hat{\vec{\alpha}}$ and appears to come to the observer under an angle $\vec{\theta}$. In these situations the distances involved are large and the angles will thus be small. If D_s is the angular diameter distance from the observer to the source, D_d between the

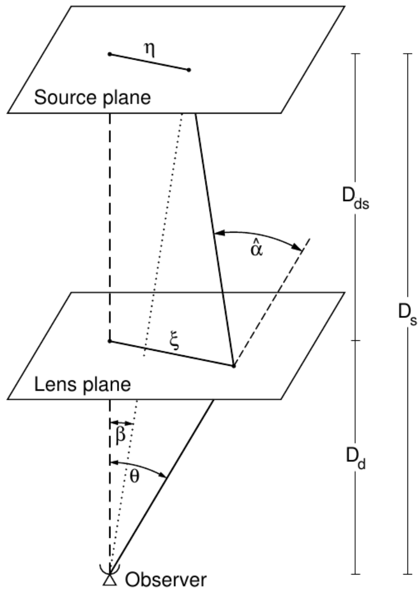


Figure 2.1 – A schematic representation of a lens system

observer and the lens and D_{ds} the angular diameter distance from the lens to the source, then by looking at the geometry it follows that

$$\vec{\eta} = \frac{D_s}{D_d}\vec{\xi} - D_{ds}\hat{\vec{\alpha}}(\vec{\xi}) \tag{2.3}$$

and

$$\vec{\eta} = D_s\vec{\beta} \qquad \vec{\xi} = D_d\vec{\theta} \tag{2.4}$$

Filling in relations 2.4 into equation 2.3 then gives the lens equation.

$$\vec{\beta} = \vec{\theta} - \frac{D_{ds}}{D_s}\hat{\vec{\alpha}}(D_d\vec{\theta}) \tag{2.5}$$

By scaling the deflection angle $\hat{\vec{\alpha}}$ over the distances

$$\vec{\alpha} \equiv \frac{D_{ds}}{D_s}\hat{\vec{\alpha}}(D_d\vec{\theta}) \tag{2.6}$$

equation 2.5 can be simplified into

$$\vec{\beta} = \vec{\theta} - \vec{\alpha}\left(\vec{\theta}\right) \tag{2.7}$$

This is the final form of the lens equation. In general there is more than one solution of the lens equation. Therefore multiple images of the same source are able to form.
The deflection angle $\vec{\alpha}$ can also be written as the gradient of a dimensionless potential.

$$\vec{\alpha} \equiv \vec{\nabla}\psi \tag{2.8}$$

We will call ψ the lens potential and it describes the effect of the lens on the deflection of light rays coming from the source. In our simulations we will perturb this potential with fluctuations that are representative of small-scale structure in the lens. In order to get to a description of these fluctations we will need some knowledge of Fourier theory, which is what will be described in the next section.

2.2 Fourier Theory

In order to quantify the scales that are involved we use the Fourier transform. In two dimensions it is defined by

$$F(\vec{k}) = \int d\vec{x} f(\vec{x}) e^{-i\vec{k}\cdot\vec{x}} \tag{2.9}$$

The function $F(\vec{k})$ denotes the Fourier transform of $f(\vec{x})$. It breaks a function into waves with wavenumbers $\vec{k}$ and amplitudes $F\left(\vec{k}\right)$, which combined would reproduce the original. It is commonly used for sound, where the Fourier transform finds the frequencies the signal is composed of. To give an example, the Fourier transform of a sinusoidal function, or a single tone, would be a delta-function at the frequency of the wave. The space spanned by the $\vec{k}$ vector will be called Fourier space. Here, a k-value represents a certain scale λ in real space, where $k = \frac{2\pi}{\lambda}$. One should keep in mind that the highest k-values correspond to the lowest scales and vice versa. A similar relation exists to transform back to real space and is called the Inverse Fourier transform.

$$f(\vec{x}) = \int \frac{d\vec{k}}{(2\pi)^2} \hat{f}(\vec{k}) e^{i\vec{k}\cdot\vec{x}} \tag{2.10}$$

Another standard result from the Fourier transform is a block wave which transforms into a sinc function. The important result that is required for this research however is the transform of a Gaussian. Take a function f(x) of the form $f(x) \propto e^{-ax^2}$ and insert it into the Fourier transform. Here a one dimensional example is used, but it does not matter for the end result because the Fourier transform integrates over each dimension separately.

$$\begin{aligned}
F(k) &= \int_{-\infty}^{\infty} dx\, f(x)\, e^{-i\,k\,x} \\
&= \int_{-\infty}^{\infty} dx\, e^{-ax^2}\, e^{-i\,k\,x} \\
&= \int_{-\infty}^{\infty} dx\, e^{-ax^2}\, (cos(kx) - isin(kx)) \\
&= \int_{-\infty}^{\infty} dx\, cos(kx) e^{-ax^2} - i \int_{-\infty}^{\infty} dx\, sin(kx) e^{-ax^2} \\
&= \int_{-\infty}^{\infty} dx\, cos(kx) e^{-ax^2} - 0 \\
&= \sqrt{\frac{\pi}{a}} e^{\frac{-k^2}{4a}}
\end{aligned}$$

The second to last step uses the fact that the integral of an odd function over a symmetric interval equals zero. The cosine term does not suffer this fate and is given by a standard integral[9]. The conclusion is that the Fourier transform of a Gaussian function results in another Gaussian function.

Another important theorem is Parseval's theorem.

$$\int |f(\vec{x})|^2\, d\vec{x} = \int \left|F(\vec{k})\right|^2 d\vec{k} \tag{2.11}$$

It relates the power of a function in real space to the power in Fourier space. Equations (2.9), (2.10) and (2.11) will be used for the creation of the fluctuations in the lens potential. In chapter 3 the implementation into a numerical code is discussed, where the relations are changed from continuous functions to discrete ones.

2.3 A Description For Fluctuations

In order to add potential fluctuations to the simulation of a lens, we need some way to describe them and model the appropriate scales. We model the fluctuations using a Gaussian random field. In its most basic definition, a random field is simply a grid filled with random numbers following a certain distribution. For the potential fluctuations we assume a Gaussian distribution. Because we are interested in the overal scales on which the potential fluctuations manifest themselves and not on the exact shape of it, we need some representation of this. The first thing that comes to mind is to work from Fourier space, because there the scales are actually represented. A useful tool to then quantify the scales is called the 'Power Spectrum'. It is defined as the absolute value of the field in Fourier space squared (equation (2.12)).

$$P(\vec{k}) \equiv F^*(\vec{k})F(\vec{k}) = \left|F(\vec{k})\right|^2 \tag{2.12}$$

The power spectrum describes the amount by which scales are present. For a Gaussian random field, it is related to the auto-correlation function ξ through a Fourier transform[10].

$$\xi(\vec{x}) = \int \frac{d\vec{k}}{(2\pi)^2} P(\vec{k}) e^{-i\vec{k}\cdot\vec{x}} \tag{2.13}$$

ξ determines the correlation between two points in an image and therefore describes the statistics of the entire fluctuation field. So if the power spectrum is known, so is the auto-correlation function and thus the entire random field can be created. For the creation of the potential fluctuations, the power spectrum will determine the standard deviation of the Gaussian random numbers that will be generated in Fourier space. Then by Fourier transforming the Fourier space information, one gets another image with values that are normally distributed due to the properties of the transform shown earlier. In preparation to this research, from numerical simulations, the typical powerspectra of fluctuations is described as $P \propto k^{-n}$ where n is assumed to be either -4 or -6. Normalization of the powerspectra to a specific fluctuation variance is acquired through Parseval's theorem, which will be shown in the next chapter.

2.4 Observational Parameters

Adding the potential fluctuations to the lens potential gives us a representation of a lensing event we would find in real observations. To look at the alterations the fluctuations make to an image that would be observed if there would be no noise and no potential fluctuations, we subtract such a smooth model from the simulations. These acquired residuals then should contain all the information about the random noise and deviations from the unperturbed potential simulation. We again use a power spectrum for extracting the scale information. By changing several parameters (noise level, source size and potential fluctuation size) we try to constrain the requirements for being able to extract such a power spectrum from real data compared to a smooth model of the lens system.

How all the above discussed aspects were implemented is described next, where we go through the numerical code that was used for the simulations.

Chapter 3

Numerical Implementation

When equations don't look very complicated in the continuous case, chances are that it will be a lot harder to implement them in a working numerical code. The obvious difficulty is that everything must be calculated using discrete methods. This requires a change in the most important equations like the definition of the Fourier transform and Parseval's theorem. For this research, a code written by Koopmans [11] (2005) was used to simulate a lensing event. New scripts were then created in PYTHON for adding fluctuations to the lens potential and for extracting a power spectrum from a simulation. All the codes that were used can be found in appendices B through F and in this chapter we will give a description of the algorithms involved.

3.1 A Brief Overview of the Simulation Code

To begin we will give a short summary of the code used for the simulations written by Koopmans [11]. It simulates an Einstein ring lensed image of an elliptical source galaxy on a 4x4 arcsecond image. It requires a separate file containing a point spread function (PSF) for specific instruments. For this research a Hubble Space Telescope PSF was provided beforehand, but in principle the simulation can also run for other telescopes. The simulation consists of three components, namely the source, the lens and a lensed image of the source.

The source brightness is described by an exponential function with a peak brightness of 100 intensity units and a size parameter σ_{src}. The units of the flux can be arbitrarily chosen. There are two ways in which to alter the size of the source. The first is to keep the same intensity in the middle of the object and broadening or narrowing the rest of the function. A second option is to give every source the same integrated intesity, which decreases the central intensity when the source size is increased. Both models were included into the code, but in the end the latter was not used because sources with the same peak brightness make it easier to compare different signal to noise levels, whereas increasing the size of a flux normalized model would significantly decrease the brightness at the center.

The lens galaxy is also modeled as an ellipsoid described by the Singular Isothermal Ellipsoid, or SIE for short from Kormann et al. [12] (1994). It first determines the lens potential, which is the part where for this research potential fluctuations were added and then it calulates the deflection angle at every point and finally creating the image of the source galaxy lensed by the SIE lens on a grid with 80×80 pixels and a size of $4'' \times 4''$. The code also uses a second smaller lens object, but for this project we set its strength to zero. The final stage of the image creation is to artificially add noise using a normal distribution random number generator. The maximum level of the noise is also one of the parameters that was varied. An example of an end product from the code is given in figure 3.1.

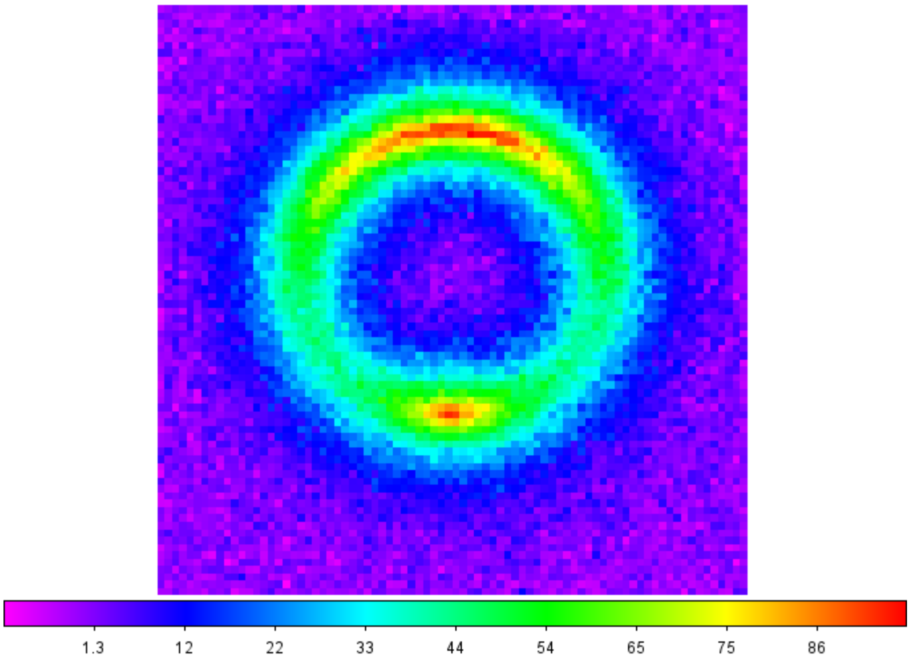


Figure 3.1 – Example from the lensing code. A source of size $\sigma_{src} = 0.25$ was used. Gaussian noise with a maximum level of 3.0 intensity units was added to the final image

Only minor adjustments were made to the script in order for it to use newly created codes to incorporate potential fluctuations in the lens galaxy model. A more detailed description can be found in the original article written about the code [11]. The full script is listed in Appendix B.

3.2 Modeling Fluctuations on the Lens Potential

Substructure in the lens object was modeled using fluctuations on the lens potential. These are described by the Gaussian random field, whose scales are characterized by a power spectrum. To generate a random field we started from Fourier space by filling in a grid with normally distributed random numbers and then Fourier transforming it to get the final image that represents the actual structure in the lens potential and which can then be added to it. Because we are talking about real objects, the actual creation of the grid requires the implementation of some special conditions which will be derived next.

Creating a Grid in Fourier Space

The first challenge that is faced when trying to simulate the fluctuations is how to start with a grid in Fourier space that produces a real image after it is inverse Fourier transformed. An inherent property of the definition of the Fourier transform is that it contains complex numbers, which in most cases would result in a complex function. It should however be possible to create real images, but that takes different handling of the function in Fourier space. A second condition that is imposed on the field is that it has to average out to zero, because the fluctuations should only represent the small-scale structure in the lens and not add to the total mass. How these conditions are represented in the grid in Fourier space can be derived by looking at the definition of the Discrete Fourier transform implemented by the NUMPY package in PYTHON [13].

$$F_{kl} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_{mn} \cdot e^{-2\pi i \left(\frac{mk}{M} + \frac{nl}{N}\right)} \tag{3.1}$$

$$f_{mn} = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} F_{kl} \cdot e^{-2\pi i \left(\frac{mk}{M} + \frac{nl}{N}\right)} \tag{3.2}$$

Here F_{kl} is the Fourier transform of f_{mn} and this definition is valid for a rectangular M×N grid. For this simulation we will assume that M and N are equal, but the code can handle alterations to the shape of the grid without any problems. Note that this definition implies the use of normal frequencies in stead of angular frequencies (or wavenumbers) and therefore all the Fourier components in the code use the parameter $l = \frac{k}{2\pi}$, which is what therefore will be adopted in the rest of this chapter. PYTHON and its modules have a slightly unnatural way of going through a matrix, in the sense that it does not recognize the center of the matrix as the origin of the axes and in stead starts at the top left corner. The Fourier transform module then adds to this difficulty by adopting its own standard order to place the frequencies along the axes. This should all be taken into account when filling in the Fourier plane, but first we will derive a couple of special conditions that will impose the creation of a real image after the transformation.

To look at wat will happen on the other side of the grid, one only has to change the coordinates in the Fourier transform to the following[14]

$$k \rightarrow M - k \quad and \quad l \rightarrow N - l$$

Then the transform changes into

$$\begin{aligned} F_{M-k,N-l} &= \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_{mn} \cdot e^{-2\pi i \left(m - \frac{mk}{M} + n - \frac{nl}{N}\right)} \\ &= \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_{mn} \cdot e^{+2\pi i \left(\frac{mk}{M} + \frac{nl}{N}\right)} \cdot e^{-2\pi i (m+n)} \end{aligned}$$

The last term is equal to unity, because $e^{-2\pi i \cdot m} = e^{-2\pi i \cdot n} = 1$ for integer m and n. In the above equation f_{mn} represents the fluctuations in normal space and thus has only real values (imaginary fluctuations are not very useful physically for this model). Therefore the statement can be reduced to

$$F_{M-k,N-l} = F^*_{k,l} \tag{3.3}$$

This is identical to saying $F(l) = F^*(-l)$ in the continuous case. So there is a cross-correlation between points in the grid and thus only half of it has to be generated in order to be able to fill the entire grid. On the lines on the grid where either $k = 0$ or $l = 0$ this symmetry turns into

$$F_{0,N-l} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_{mn} \cdot e^{2\pi i \frac{nl}{N}} = F^*_{0,l} \tag{3.4}$$

$$F_{M-k,0} = F^*_{k,0} \tag{3.5}$$

The three conditions given in equations (3.3), (3.4) and (3.5) are the basic requirements of the grid in Fourier space that are necessary to make the grid real-valued after it has gone through a Fourier transformation. There are however four special points at the halfway points (also known as the Nyquist frequency[14]) that need to be taken care of at $k = \frac{M}{2}$ and $l = \frac{N}{2}$.

$$F_{\frac{M}{2},0} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_{mn} \cdot e^{-\pi i \cdot m} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_{mn} \cdot (-1)^{m} \tag{3.6}$$

$$F_{0,\frac{N}{2}} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_{mn} \cdot e^{-\pi i \cdot m} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_{mn} \cdot (-1)^{n} \tag{3.7}$$

$$F_{\frac{M}{2},\frac{N}{2}} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_{mn} \cdot e^{-\pi i (m+n)} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_{mn} \cdot (-1)^{(m+n)} \tag{3.8}$$

$$F_0, 0 = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_{mn} = 0 \tag{3.9}$$

These four points in the grid are real-valued, because f_{mn} (the function in real space) is as well. Equation (3.9) describes an even more special case. For the location in Fourier space where both k and l equal zero, the value of the grid is just the sum of all the values of the field in real space. This is simply the average of the field (without a factor $\frac{1}{MN}$) and so this point can be set to equal zero. It should be noted that even though these conditions are built into the code, the resulting imaginary part of the inverse Fourier transformed grid will not be exactly zero. This is due to numerical errors like the finite number of decimals that can be stored. As long as the imaginary values are very small, one can assume that it is zero and only use the real part. If the values are not negligible, then something has gone wrong in the implementation of the conditions for a real image. For our code the imaginary values were of the order of $\sim 10^{-20}$ and therefore definitely not significant.

The derived conditions apply to every case where one wants the discrete inverse Fourier transform to result in a real-valued image but they don't depend on the physical dimensions of a required image. The simulation code creates a lens potential with a specific size of the image in arcseconds ($4'' \times 4''$). The image of the fluctuations that is added to the lens potential should then also have the same dimensions. There is a relation between the dimensions in the Fourier grid and the grid in normal space. In order to make sure that a value in Fourier space (l_x, l_y) corresponds to the right value in real space (x,y) the Fourier grid should contain these frequencies[15]:

$$\frac{-M}{2} \cdot \frac{1}{L_x} \leq l_x \leq \left(\frac{M}{2} - 1\right) \cdot \frac{1}{L_x} \qquad\qquad \frac{-N}{2} \cdot \frac{1}{L_y} \leq l_y \leq \left(\frac{N}{2} - 1\right) \cdot \frac{1}{L_y}$$

Here L_x and L_y denote the length in the x- and y-direction for the simulated image in physical units. As was mentioned before, the order of the frequencies along the axes is implemented differently by NUMPY. It places the $l = 0$ component first after which the positive frequencies follow and the negative ones come last. Figure 3.2 gives a graphical summary of how the grid was constructed.

The Gaussian Random Field

Once the Fourierplane itself has the right dimensions and follows the correct conditions, the grid can be filled in to create the fluctuations. For the Gaussian random field a power spectrum is assumed of the following form.

$$P(k) = A \cdot l^{-n} \tag{3.10}$$

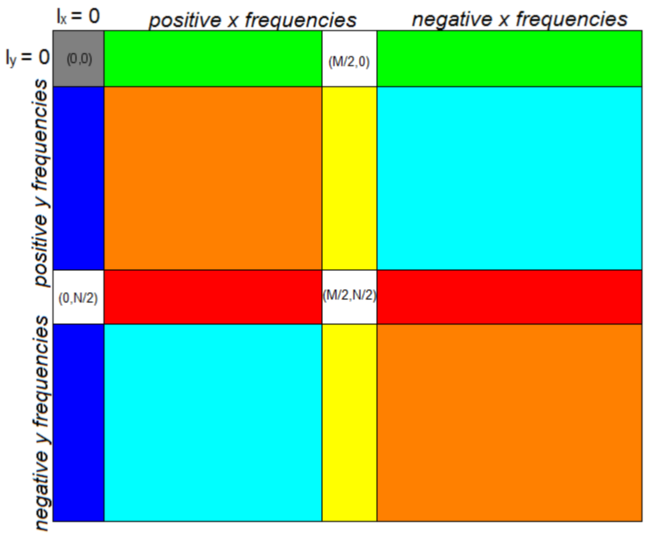


Figure 3.2 – Graphical representation of how the grid was built in Fourier space. Planes of the same color are each other's mirror image and complex conjugate. The three squares that are coloured white represent the three real valued points of the grid (equations (3.6), (3.7) and (3.8)). The gray area top left is the point equal to zero, representing the mean of the field (see equation (3.9)).

The n gives the slope of the spectrum and in this project two different values for it will be tested, namely -4 and -6. One problem that arrises with this kind of power spectrum is the sharp increase at low l-values. For l=0 it becomes infinitely large, which would signify infinite power for the largest scales and is not physical. Therefore it is set to zero in the code. Another reason to ignore this frequency is because it represents infinite scale and that is something that cannot be measured anyway in a finite field of view. The parameter A is a normalization constant and is related to the variance of the density fluctuations which will be shown later.

The random field is created by using Gaussian random numbers at each point in the grid, where the square root of the power spectrum at that frequency gives the standard deviation of the distribution. A fast and easy to program method for generating normally distributed numbers is to use the polar form of the Box-Muller transform[16]. This generates two independent Gaussian random numbers. The complex values in the Fourier space grid will be of the form $F(l) = f_1(l) + i\, f_2(l)$ where f_1 and f_2 are the numbers generated from the Box-Muller transform. It works by generating uniformly distributed random numbers u and v with values between -1 and 1. The transform from a uniform to a Gaussian distribution is then made by the following relations.

$$f_1 = u\sqrt{\frac{-2\ln(s)}{s}}$$
$$f_2 = v\sqrt{\frac{-2\ln(s)}{s}}$$

Where $s = u^2 + v^2 \leq 1$. These relations give a distribution with unit variance, but for the fluctuations a non-unit variance is necessary. To solve this problem one only has to multiply the numbers by the preferred standard deviation (or the square root of the power spectrum value in this case) and get[17]

$$f_1(l) = u\,\sqrt{P(l)}\,\sqrt{\frac{-2\ln(s)}{s}} \tag{3.11}$$

$$f_2(l) = v\,\sqrt{P(l)}\,\sqrt{\frac{-2\ln(s)}{s}} \tag{3.12}$$

Where $f_1(l)$ is used as the real part of the complex number F(l) and $f_2(l)$ as the imaginary part. In order to get a certain size scale of the potential fluctuations, the power spectrum still needs to be normalized. The constant A is related to the variance of the fluctuations through Parseval's theorem. Because of the discrete Fourier transform used in the code, the theorem takes on the form given below[18].

$$\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} |f_{mn}|^2 = \frac{1}{MN}\sum_{k=-\frac{M}{2}}^{\frac{M}{2}-1}\sum_{l=-\frac{N}{2}}^{\frac{N}{2}-1} |F_{kl}|^2 \tag{3.13}$$

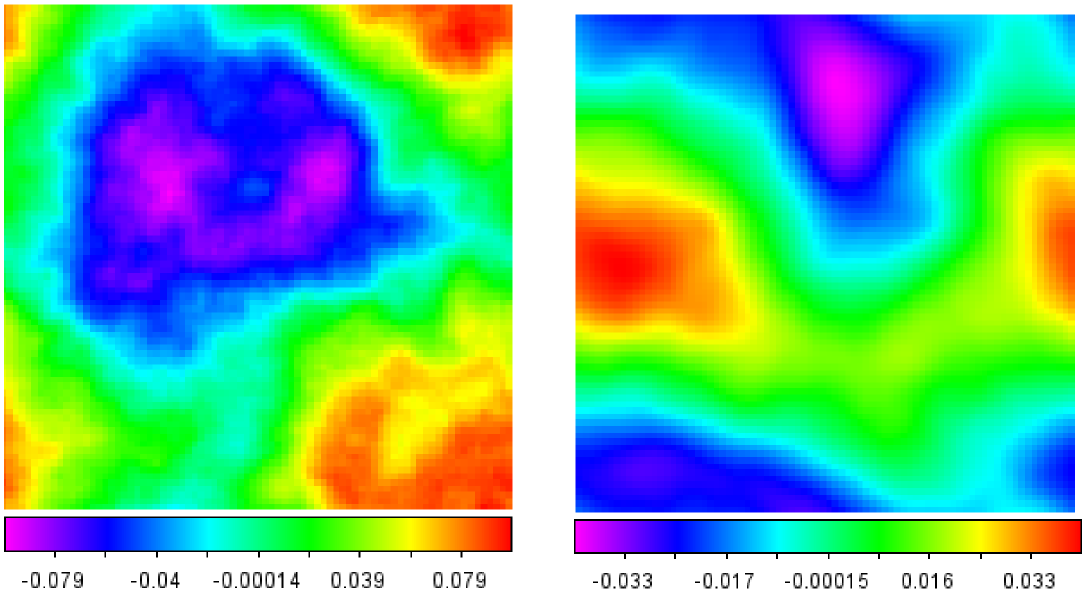
The same relation should then also hold for the variances where we will now call $MN \equiv N_{pix}$ which gives the number of pixels of the image. Using Parseval's theorem, we can derive the relation between the variance of the potential fluctuations in real space (σ_{fluct}) and the power spectrum as follows.

$$\begin{aligned}
\sum_{m,n}\sigma^2_{mn} &= \frac{1}{N_{pix}}\sum_{k,l}\sigma^2_{kl} \\
\Rightarrow N_{pix}\cdot\sigma^2_{fluct} &= \frac{1}{N_{pix}}\sum_{k,l}\sigma^2_{kl} \\
&= \frac{1}{N_{pix}}\sum_{k,l}P_{kl} \\
\Rightarrow \sigma^2_{fluct} &= \frac{1}{N^2_{pix}}\sum_{k,l}P_{kl}
\end{aligned}$$

In the second line we have used the fact that the variance in real space should always have the specified value that is desired. Solving for A from the power spectrum then gives the correct normalization.

$$A = \frac{\sigma^2_{fluct}N^2_{pix}}{2\sum l^{-n}} \tag{3.14}$$

An extra factor of two is added in the denominator to get the correct outcome. This is needed because half of the grid is generated and the rest of it is taken to be the be the complex conjugate of the first part. In Fourier space however, a point and its complex conjugate are not independent. The result is a factor two increase of the variance for which we need a correction. Also take note that the power spectrum is set to zero for l = 0. Once the complex random numbers are correctly added to the grid and taking care off relations (3.3) through (3.9), the fluctuations on the lens potential are obtained through an inverse discrete Fourier transform. It is then added to the lens potential and will be used in the simulations. The implementation in PYTHON can be found in Appendix C and figure 3.3 gives two realisations for different power spectra of the fluctuations. In figure 3.4 we show an example of a lensed image with noise and potential fluctuations added to the lens.



(a) $P(l) = A \cdot l^{-4}$ (b) $P(l) = A \cdot l^{-6}$

Figure 3.3 – Examples of Gaussian random fields for two different power spectra for a fluctation variance $\sigma^2_{fluct} = 10^{-3}$. The realisation with a steeper spectrum (right) clearly has less structure on small scales.

If all goes well, the root mean square of the pixel values of the random field should be close to the square root of the variance in the power spectrum in the normalization constant. A single realisation of a field will not yet give a real normal distribution. After multiple generations, they all together should give a Gaussian due to the Central Limit theorem. To check the distribution of multiple fields, a script was written, which can be found in appendix F. A histogram of the distribution of 100 random fields is given in figure 3.5 for $\sigma^2_{fluct} = 10^{-5}$.

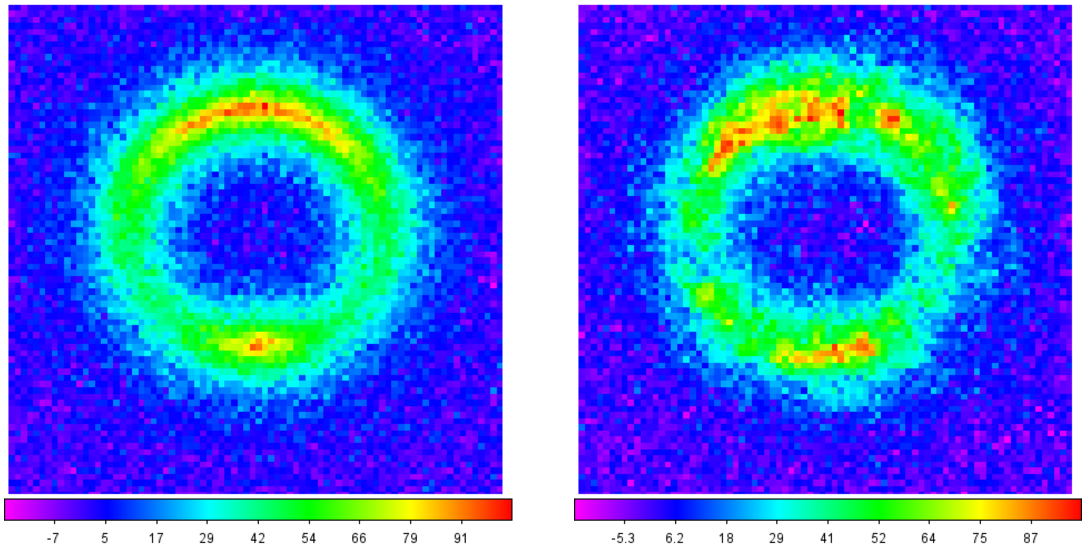


Figure 3.4 – Lensed images with no potential fluctuations (*left*) and with fluctuation size $\sigma^2_{fluct} = 10^{-3}$ (*right*). The source size is $\sigma_{src} = 0.25$ and the images have a noise level of 5.0 intensity units. The fluctuation power spectrum used here has a slope of -4.

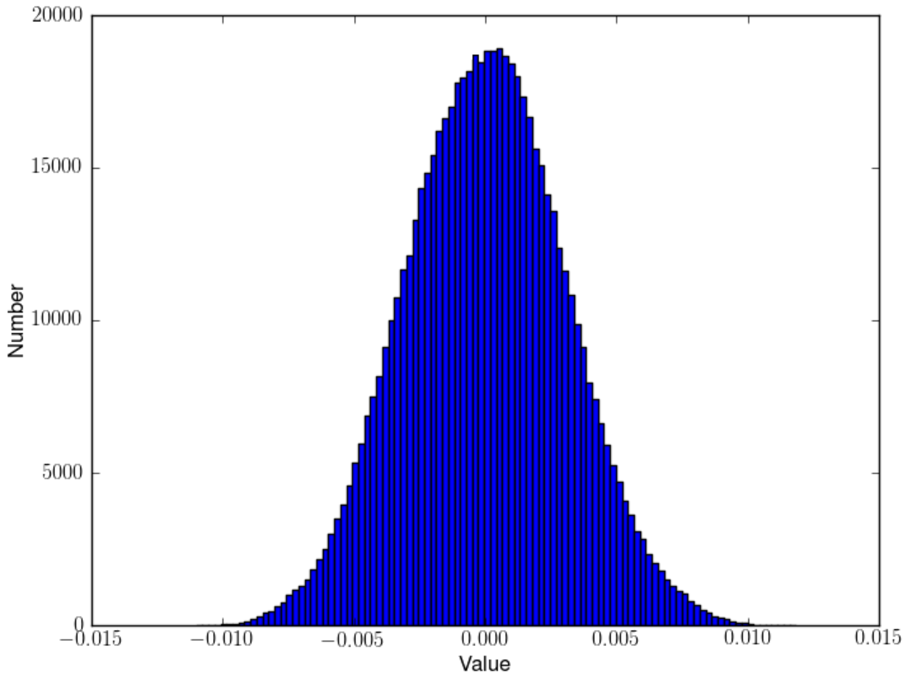
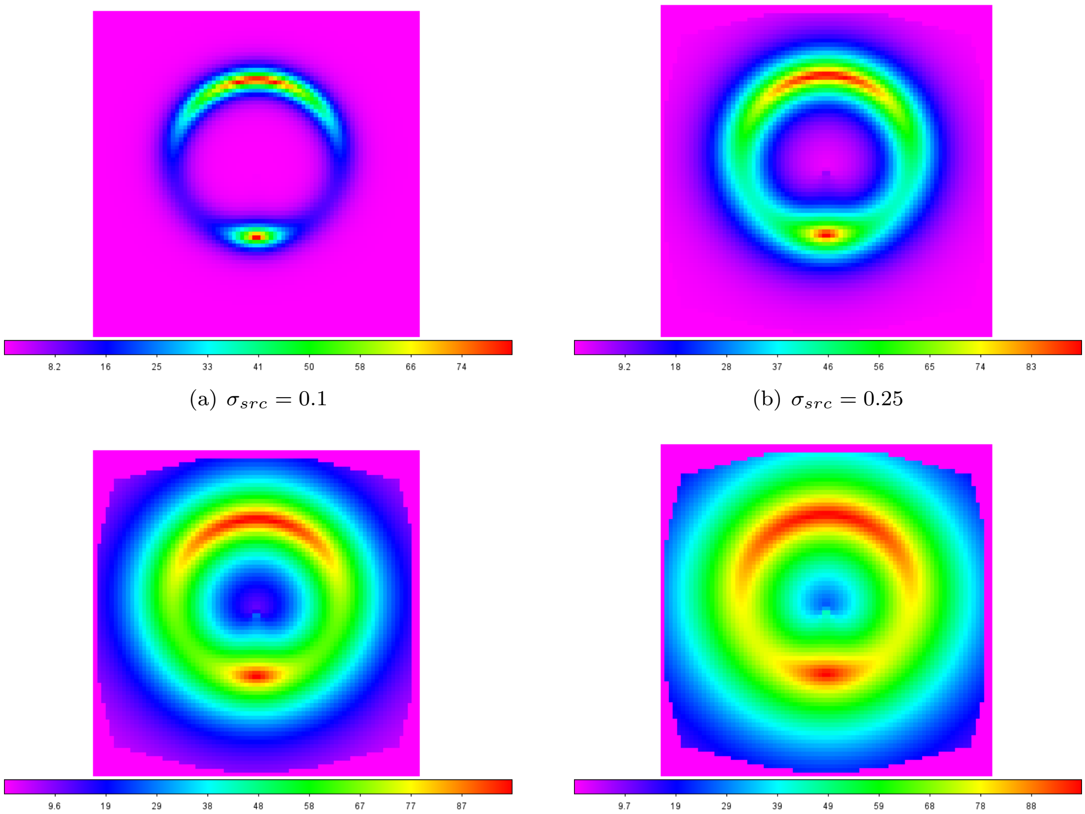


Figure 3.5 – Distribution of 100 Gaussian random fields, with $\sigma^2_{fluct} = 10^{-5}$. The variance of the distribution should equal σ^2_{fluct}

3.3 Getting Information From Residuals

After the simulation with potential fluctuations is performed, we would like to see if it is possible to get some information back on the fluctuations. For this we create residuals by subtracting a smooth model of the lens system from the simulation data. The smooth model is a simulation where no noise and no potential fluctuations were added. The lensed images produced by the smooth model for four separate source sizes are given in figure 3.6.



(a) $\sigma_{src} = 0.1$ (b) $\sigma_{src} = 0.25$

(c) $\sigma_{src} = 0.5$ (d) $\sigma_{src} = 0.8$

Figure 3.6 – The four lens models for different source sizes

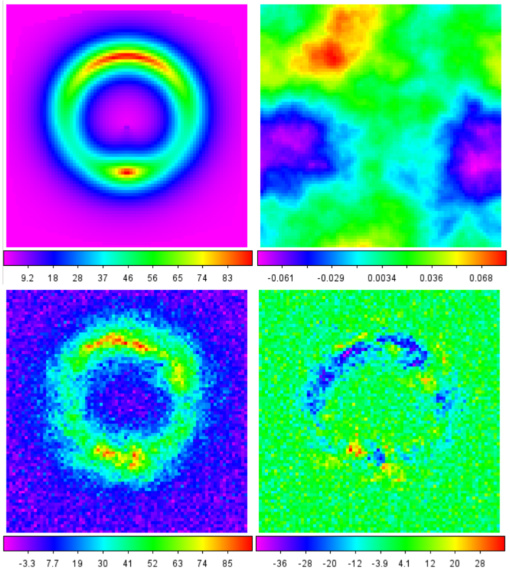
The resulting residuals can then be Fourier transformed, and from the absolute value squared we obtain their power spectrum by dividing the image into ten radial bins. In each frequency bin the mean value is then taken to get the power at that scale. This is done for a hundred potential fluctuation generations, which will also provide a spread at each scale in the powerspectrum, representing the error if one would only extract the powerspectrum from a single observation. The error in each bin j is calculated using the root mean square deviation from the mean value.

$$\sigma_j^2 = \text{rms}_j^2 = \frac{\sum_{i=1}^{N} (P_{ij} - < P >_j)^2}{N - 1} \tag{3.15}$$

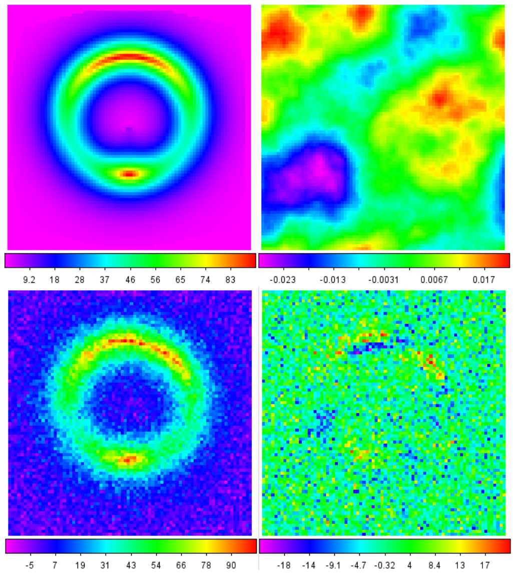
The more lensing events we can find, the more the power spectrum can be constrained with increasingly smaller errorbars. The error for N observations is determined by dividing the error for a single measurement by $\sqrt{N}$. The reader is reffered to Appendices D and E for the code used to create residuals and determine power spectra and for the plotting of some of the results.

To get some idea of the different stages that are excecuted and how the different potential fluctuation scales influence the lensed image and the residuals, we give a schematic overview in figure 3.7. Figure 3.8 shows the same but now for the steeper fluctuation power spectrum.

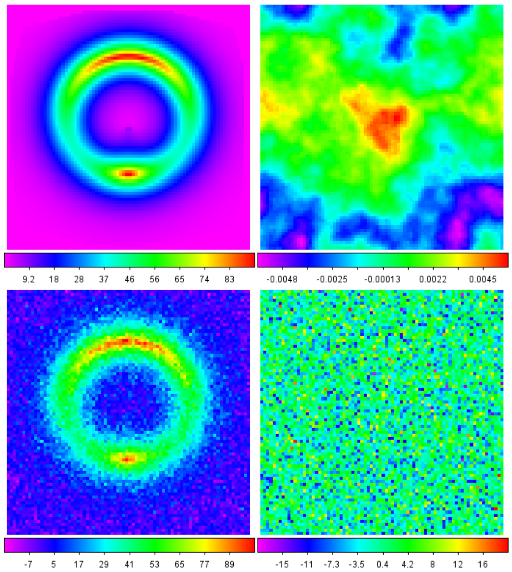
From the figures we see that a smaller σ^2_{fluct} gives the random fields more structure at smaller scales and also reduces the visibillity of any remnants of the potential fluctuations in the residuals image. These become completely dominated by noise for the lowest σ^2_{fluct} values.



(a) $\sigma^2_{fluct} = 10^{-3}$

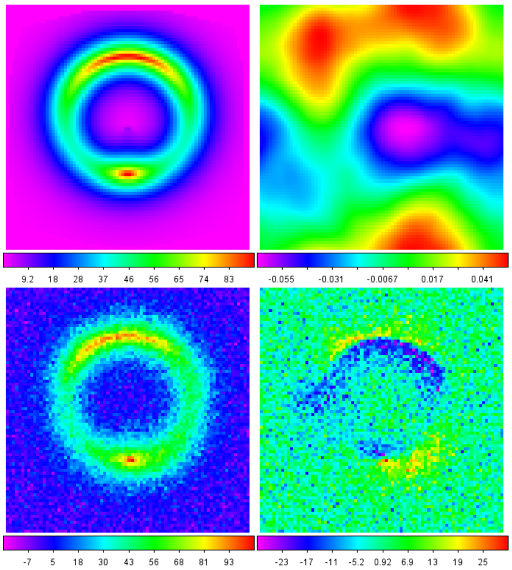


(b) $\sigma^2_{fluct} = 10^{-4}$

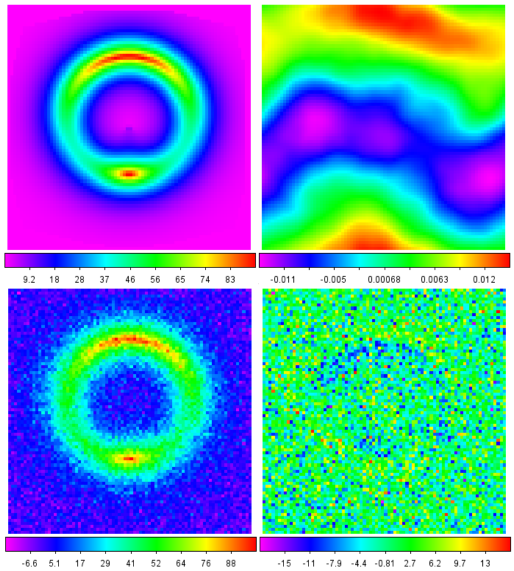


(c) $\sigma^2_{fluct} = 10^{-5}$

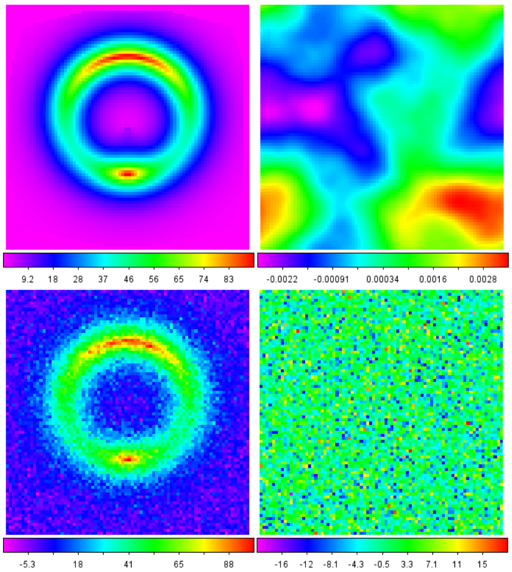
Figure 3.7 – Schematic overview of the different codes for a fluctuation power spectrum of $P(l) \sim l^{-4}$ and a source size of $\sigma_{src} = 0.25$. Each panel is for different fluctuation scales σ^2_{fluct}. In all panels the top left image give a lensed image without noise or potential fluctuations. The top right image gives the Gaussian random field that is added to the lens potential. Then the bottom left image give a lensed image with the potential fluctuations included and a noise level of 5.0 intensity units. Finally the bottom right image then gives the residuals from subtracting the top left image from the bottom left one.



(a) $\sigma^2_{fluct} = 10^{-3}$



(b) $\sigma^2_{fluct} = 10^{-4}$



(c) $\sigma^2_{fluct} = 10^{-5}$

Figure 3.8 – Schematic overview of the different codes for a fluctuation power spectrum of $P(l) \sim l^{-6}$ and a source size of $\sigma_{src} = 0.25$. Each panel is for different fluctuation scales σ^2_{fluct}. In all panels the top left image give a lensed image without noise or potential fluctuations. The top right image gives the Gaussian random field that is added to the lens potential. Then the bottom left image give a lensed image with the potential fluctuations included and a noise level of 5.0 intensity units. Finally the bottom right image then gives the residuals from subtracting the top left image from the bottom left one.

Chapter 4

The Simulation results

In this chapter we examine the result of the simulations as described previously. We first check the residuals that arise from an original fluctuation power spectrum with a slope of -4. Given in figure 4.1 are the mean power spectra of 100 residuals for a source with $\sigma_{src} = 0.1$, the smallest source that was modeled. There are several effects that will be varied. Namely the effect of noise on the measured power spectrum for several potential fluctuation scales. Later we will discuss how the shape of the spectra changes with an increase in the size of the source and finish with a steeper input power spectrum.

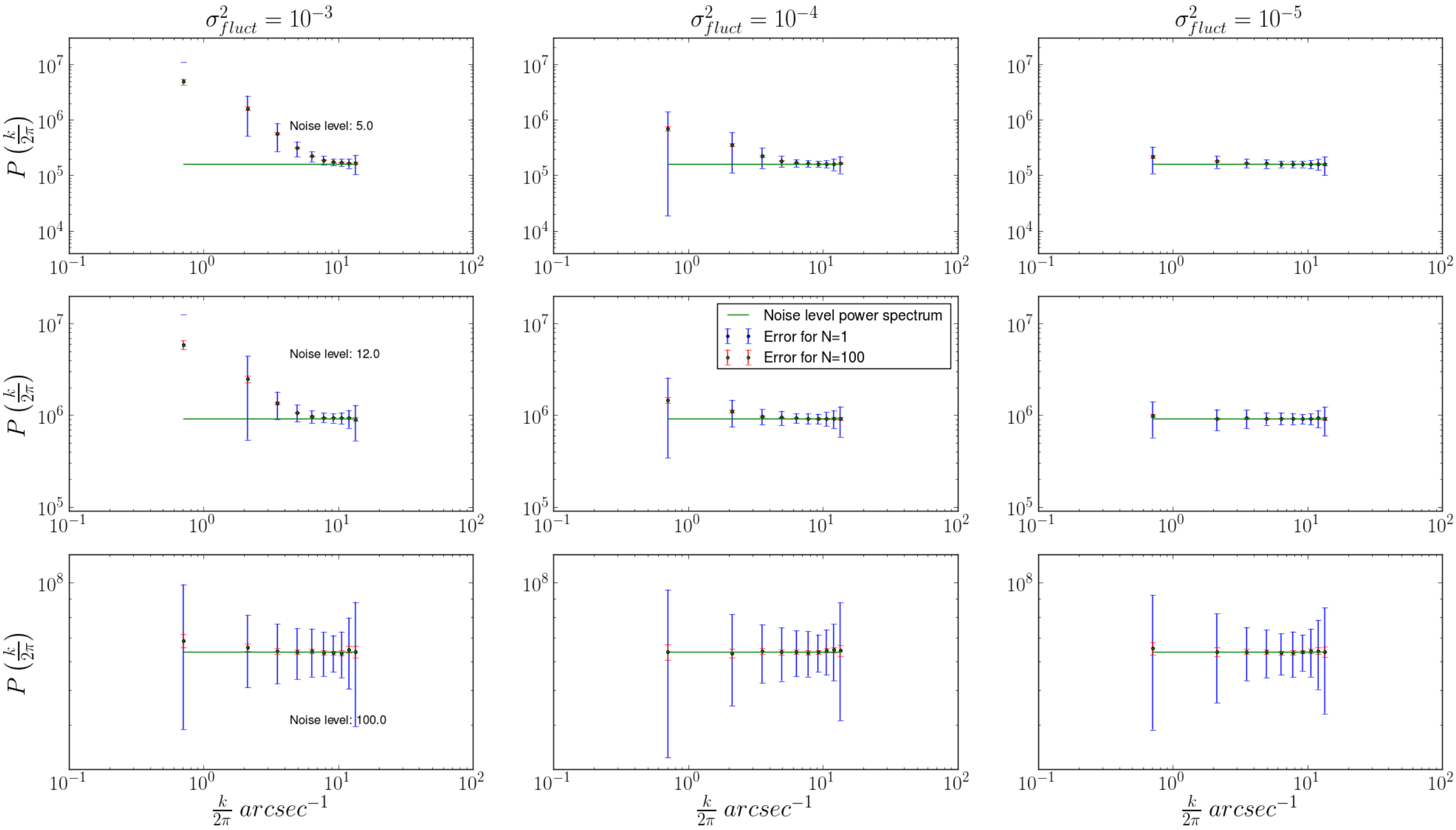


Figure 4.1 – Power spectra for a source size $\sigma_{src} = 0.1$. Each row represents a different noise level, from top to bottom 5.0, 12.0 and 100.0 respectively. The horizontal green line gives the power spectrum expected for that noise level. Every column has the same fluctuation sizes, from left to right $\sigma^2_{flucts} = 10^{-3}, 10^{-4}, 10^{-5}$ respectively. Blue errorbars give the error for a single measurement, whereas the red errorbars would be the error after 100 observations. If only an upper limit to the error is given (a small horizontal blue line), the lower limit extends to negative values and thus cannot be plot on a logarithmic scale. The fluctuation inputspectrum was $P(l) \propto l^{-4}$.

In some of the datapoints only the upper limit of the error is given. This is due to the lower limit extending to negative values, which can not be plot on a logarithmic scale.
The errorbars in the plots show an increase for the largest scales and the smallest scales. The latter will be dominated by errors from the noise. Large scales however have uncertainty due to sample variance. This is the error that arises due to the finite number of measurements performed and should decrease with a larger number of simulations. This same trend will be visible in all the power spectra that were extracted.

4.1 Effects of Noise

In general every observation will have some influence from noise. Together with the spatial resolution of the telescope noise affects the accuracy with which the power spectra can be measured. The stronger it fluctuates, the harder it will be to get accurate results. As long as the noise is truly random, it will converge to a contstant powerspectrum with an amplitude of $N_{pix} \times (\sigma_{noise})^2$. Simulations that only contained noise did indeed show an almost constant line with this amplitude. This noise power spectrum can therefore be added to the plots and is given by the green straight line.

In figure 4.1, each row has the same noise level, increasing downwards and all the plots in a single column have the same potential fluctuation size, decreasing rightwards. If the power spectra agree with that expected from the noise within the measurement error, it will be hard to measure them. As can be seen from the plots, for the highest noise level, where the signal to noise ratio is lower than unity at every point of the lensed image, obtaining a residual power spectrum will be near impossible. Also, the smaller the fluctuations are, the closer the spectrum reaches the noise level which is especially problematic for potential fluctuations of the order $\sigma^2_{fluct} = 10^{-5}$. For the higher σ^2_{fluct} values however, noise poses less of a problem at the intermediate and large scales. The higher k-values do lie closer to the noise level and the measurement there can be dominated by noise. Getting a more accurate power spectrum can however be achieved, but multiple observations will be required.

4.2 Different Source Sizes

In order to get a better understanding of how changing some of the parameters in the simulations would affect the residuals, we generated spectra for several situations. One of the options is to alter the size of the source. We tried the four variations already meantioned in section 3.3. The results for the largest source can be found in figure 4.2, whereas the ones for the two intermediate sized source galaxies are given in appendixA.

Overall there does not seem to be any significant change to the shape of the residual power spectra. One might argue that for the smaller sources the spectrum starts to flatten somewhat at the largest scales, but the effect falls within the errorbars for a single measurement. The errors can be reduced by measuring the power spectra from more lensing events, represented by the red errorbars. This however would assume the same level of potential fluctuations for every lens observed and that does not necessarily have to be true. A larger source does (but only slightly) increase the amplitude of the power spectra, which might just help raising it above the noise level.

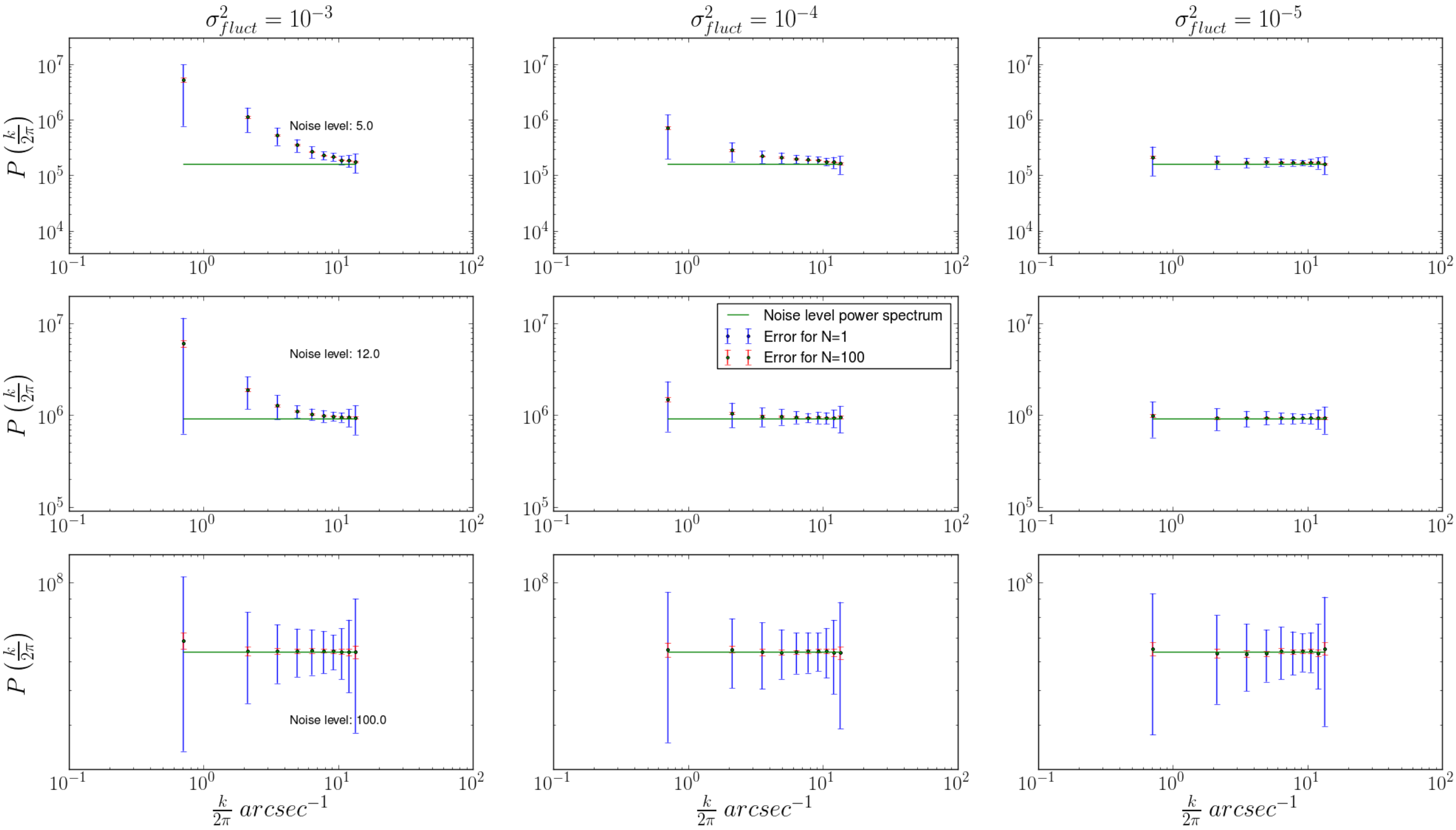


Figure 4.2 – Power spectra for a source size $\sigma_{src} = 0.8$. The rows represent different noise levels, from top to bottom 5.0, 12.0 and 100.0 respectively. The horizontal green line gives the power spectrum expected for that noise level. Every column has the same fluctuation sizes, from left to right $\sigma^2_{flucts} = 10^{-3}, 10^{-4}, 10^{-5}$ respectively. Blue errorbars give the error for a single measurement, whereas the red errorbars would be the error after 100 observations. The fluctuation inputspectrum was $P(l) \propto l^{-4}$.

4.3 Steeper Input Spectrum

The last alteration that was attempted was to let the potential fluctuations in the lens be described by a power spectrum with a steeper slope. A steeper spectrum drops faster at high k-values and therefore basically cuts off small scale density fluctuations. This was also visible in the example of random fields given in figure 3.3. The k^{-6} field there is much smoother. This should definitely influence residuals that we would get. In this section we again give the results for both $\sigma_{src} = 0.1$ and $\sigma_{src} = 0.8$, but the ones for intermediate values can be found in appendix A.
Here we see a flattening of the spectra at the high k end of the spectra. This directly follows the absence of the smaller scales due to the steepening of the input power spectrum. Now in stead of structure, the noise dominates at these scales. Therefore fluctuations with a larger slope power spectrum will give more problems with high noise levels. On the other hand, because they are larger, the small scale end of the power spectrum would probably not contain a lot information about the potential fluctuations, other than giving an idea of the mininum sizes of fluctuations.

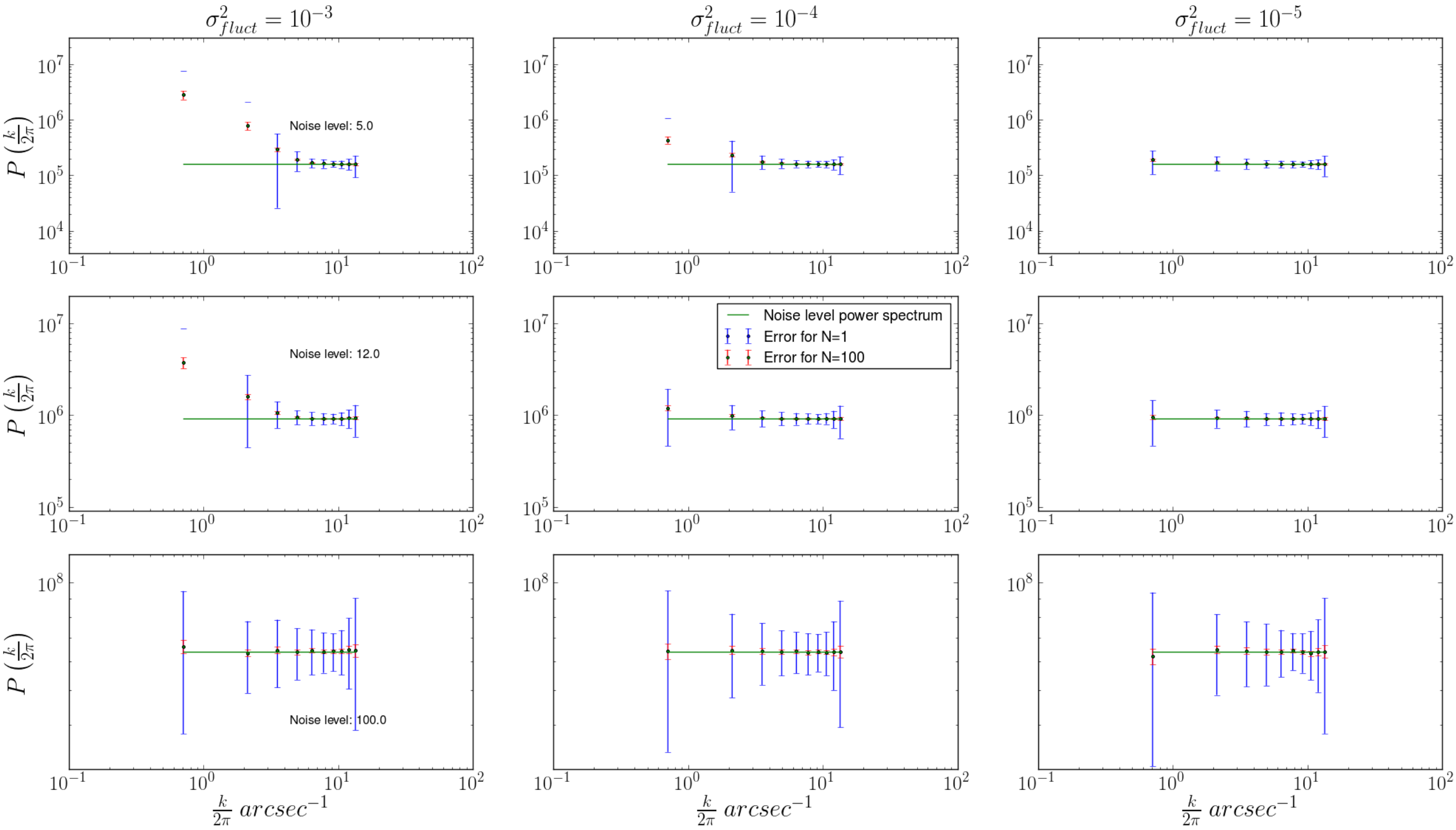


Figure 4.3 – Power spectra with a source size $\sigma_{src} = 0.1$. The rows represent different noise levels, from top to bottom 5.0, 12.0 and 100.0 respectively. The horizontal green line gives the power spectrum expected for that noise level. Every column has the same fluctuation sizes, from left to right $\sigma^2_{flucts} = 10^{-3}, 10^{-4}, 10^{-5}$ respectively. Blue errorbars give the error for a single measurement, whereas the red errorbars would be the error after 100 observations. If only an upper limit to the error is given (a small horizontal blue line), the lower limit extends to negative values and thus cannot be plot on a logarithmic scale. The fluctuation inputspectrum was $P(l) \propto l^{-6}$.

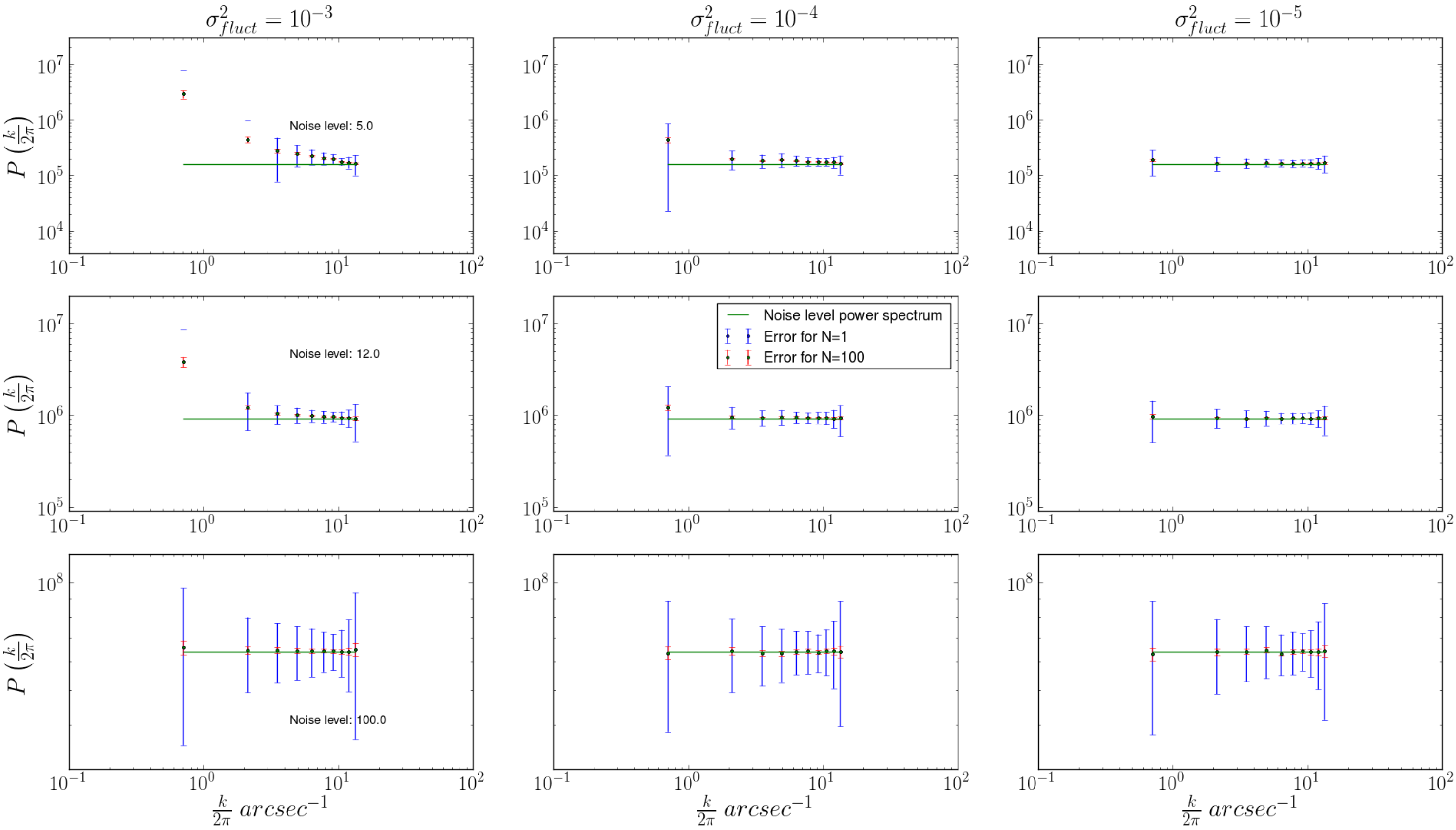


Figure 4.4 – Power spectra with a source size $\sigma_{src} = 0.8$. The rows represent different noise levels, from top to bottom 5.0, 12.0 and 100.0 respectively. The horizontal green line gives the power spectrum expected for that noise level. Every column has the same fluctuation sizes, from left to right $\sigma^2_{flucts} = 10^{-3}, 10^{-4}, 10^{-5}$ respectively. Blue errorbars give the error for a single measurement, whereas the red errorbars would be the error after 100 observations. If only an upper limit to the error is given (a small horizontal blue line), the lower limit extends to negative values and thus cannot be plot on a logarithmic scale. The fluctuation inputspectrum was $P(l) \propto l^{-6}$.

4.4 Statistics

Although the plots can be very useful to see the overall effect of the different parameters, estimating if the power spectra are able to be significantly measured requires some statistics. We adopt the χ^2 test to fit the residual power spectra to the noise level power spectrum. The χ^2 values were determined with

$$\chi^2 = \sum_{j=1}^{N} \frac{(< P >_j - P_{noise})^2}{\sigma_j^2} \tag{4.1}$$

where $< P >_j$ is the mean power spectrum at point j, P_{noise} is the noise power spectrum value and σ_j^2 is the error in the mean given by equation (3.15). The probability that the measured power spectrum is a good fit of the noise level was then determined using the CHIDIST function in the OpenOffice Calc software[19] with nine degrees of freedom (ten data points minus one and the noise power spectrum has no fittable parameters). The probability $p_{pot.fluct.}$ that the measured mean spectrum is the result of the potential fluctuations is then given by one minus the probability

$\sigma_{src} = 0.1$:						
	$\sigma_{fluct} = 10^{-3}$		$\sigma_{fluct} = 10^{-4}$		$\sigma_{fluct} = 10^{-5}$	
	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$
$\sigma_{noise} = 5.0$	9.960	$6.462 \cdot 10^{-1}$	2.276	$1.369 \cdot 10^{-2}$	$5.537 \cdot 10^{-1}$	$4.713 \cdot 10^{-5}$
$\sigma_{noise} = 12.0$	2.778	$2.755 \cdot 10^{-2}$	$6.116 \cdot 10^{-1}$	$7.199 \cdot 10^{-5}$	$3.693 \cdot 10^{-2}$	$2.974 \cdot 10^{-10}$
$\sigma_{noise} = 100.0$	$5.474 \cdot 10^{-2}$	$1.734 \cdot 10^{-9}$	$9.895 \cdot 10^{-3}$	$8.019 \cdot 10^{-13}$	$1.024 \cdot 10^{-2}$	$9.369 \cdot 10^{-13}$

$\sigma_{src} = 0.25$:						
	$\sigma_{fluct} = 10^{-3}$		$\sigma_{fluct} = 10^{-4}$		$\sigma_{fluct} = 10^{-5}$	
	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$
$\sigma_{noise} = 5.0$	$1.236 \cdot 10^{1}$	$8.062 \cdot 10^{-1}$	2.638	$2.306 \cdot 10^{-2}$	$4.131 \cdot 10^{-1}$	$1.336 \cdot 10^{-5}$
$\sigma_{noise} = 12.0$	3.975	$8.693 \cdot 10^{-2}$	$6.244 \cdot 10^{-1}$	$7.866 \cdot 10^{-5}$	$2.737 \cdot 10^{-2}$	$7.747 \cdot 10^{-11}$
$\sigma_{noise} = 100.0$	$6.423 \cdot 10^{-2}$	$3.548 \cdot 10^{-9}$	$1.189 \cdot 10^{-2}$	$1.830 \cdot 10^{-12}$	$6.641 \cdot 10^{-3}$	$1.334 \cdot 10^{-13}$

$\sigma_{src} = 0.5$:						
	$\sigma_{fluct} = 10^{-3}$		$\sigma_{fluct} = 10^{-4}$		$\sigma_{fluct} = 10^{-5}$	
	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$
$\sigma_{noise} = 5.0$	$1.823 \cdot 10^{1}$	$9.674 \cdot 10^{-1}$	3.691	$6.945 \cdot 10^{-2}$	$4.121 \cdot 10^{-1}$	$1.322 \cdot 10^{-5}$
$\sigma_{noise} = 12.0$	4.496	$1.241 \cdot 10^{-1}$	$6.886 \cdot 10^{-1}$	$1.190 \cdot 10^{-4}$	$2.495 \cdot 10^{-2}$	$5.112 \cdot 10^{-11}$
$\sigma_{noise} = 100.0$	$5.665 \cdot 10^{-2}$	$2.022 \cdot 10^{-9}$	$8.370 \cdot 10^{-3}$	$3.778 \cdot 10^{-13}$	$1.516 \cdot 10^{-2}$	$5.454 \cdot 10^{-12}$

$\sigma_{src} = 0.8$:						
	$\sigma_{fluct} = 10^{-3}$		$\sigma_{fluct} = 10^{-4}$		$\sigma_{fluct} = 10^{-5}$	
	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$
$\sigma_{noise} = 5.0$	$2.323 \cdot 10^{1}$	$9.943 \cdot 10^{-1}$	9.030	$5.655 \cdot 10^{-1}$	1.650	$4.127 \cdot 10^{-3}$
$\sigma_{noise} = 12.0$	5.885	$2.487 \cdot 10^{-1}$	$9.630 \cdot 10^{-1}$	$4.818 \cdot 10^{-4}$	$7.946 \cdot 10^{-2}$	$9.186 \cdot 10^{-9}$
$\sigma_{noise} = 100.0$	$2.469 \cdot 10^{-2}$	$4.884 \cdot 10^{-11}$	$1.049 \cdot 10^{-2}$	$1.042 \cdot 10^{-12}$	$2.024 \cdot 10^{-2}$	$2.000 \cdot 10^{-11}$

Table 4.1 – χ^2-values with respect to the noise level and the corresponding probability that the data is a measurement of the potential fluctuations for the power spectra of different sources, noises and fluctuation scales. These values are for an input power spectrum of $P(l) \sim l^{-4}$ and a single measurement of the lens system (blue errorbars).

that it is due to noise.
For all the power spectra in the results, the corresponding χ^2 values and probabilities are given in tables 4.1 and 4.2. These are the values for a single measument of the lens system (blue errorbars).

$\sigma_{src} = 0.1$:						
	$\sigma_{fluct} = 10^{-3}$		$\sigma_{fluct} = 10^{-4}$		$\sigma_{fluct} = 10^{-5}$	
	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$
$\sigma_{noise} = 5.0$	1.057	$7.050 \cdot 10^{-4}$	$5.295 \cdot 10^{-1}$	$3.893 \cdot 10^{-5}$	$1.849 \cdot 10^{-1}$	$3.933 \cdot 10^{-7}$
$\sigma_{noise} = 12.0$	$8.848 \cdot 10^{-1}$	$3.397 \cdot 10^{-4}$	$2.115 \cdot 10^{-1}$	$7.126 \cdot 10^{-7}$	$1.463 \cdot 10^{-2}$	$4.649 \cdot 10^{-12}$
$\sigma_{noise} = 100.0$	$1.794 \cdot 10^{-2}$	$1.162 \cdot 10^{-11}$	$5.327 \cdot 10^{-3}$	$4.952 \cdot 10^{-14}$	$1.774 \cdot 10^{-2}$	$1.106 \cdot 10^{-11}$

$\sigma_{src} = 0.25$:						
	$\sigma_{fluct} = 10^{-3}$		$\sigma_{fluct} = 10^{-4}$		$\sigma_{fluct} = 10^{-5}$	
	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$
$\sigma_{noise} = 5.0$	$8.212 \cdot 10^{-1}$	$2.492 \cdot 10^{-4}$	$5.463 \cdot 10^{-1}$	$4.447 \cdot 10^{-5}$	$1.698 \cdot 10^{-1}$	$2.702 \cdot 10^{-7}$
$\sigma_{noise} = 12.0$	$4.517 \cdot 10^{-1}$	$1.965 \cdot 10^{-5}$	$2.132 \cdot 10^{-1}$	$7.380 \cdot 10^{-7}$	$1.126 \cdot 10^{-2}$	$1.436 \cdot 10^{-12}$
$\sigma_{noise} = 100.0$	$1.736 \cdot 10^{-2}$	$1.003 \cdot 10^{-11}$	$1.694 \cdot 10^{-2}$	$8.987 \cdot 10^{-12}$	$6.141 \cdot 10^{-3}$	$9.381 \cdot 10^{-14}$

$\sigma_{src} = 0.5$:						
	$\sigma_{fluct} = 10^{-3}$		$\sigma_{fluct} = 10^{-4}$		$\sigma_{fluct} = 10^{-5}$	
	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$
$\sigma_{noise} = 5.0$	1.637	$4.000 \cdot 10^{-3}$	$5.770 \cdot 10^{-1}$	$5.621 \cdot 10^{-5}$	$1.296 \cdot 10^{-1}$	$8.133 \cdot 10^{-8}$
$\sigma_{noise} = 12.0$	$5.252 \cdot 10^{-1}$	$3.757 \cdot 10^{-5}$	$1.661 \cdot 10^{-1}$	$2.445 \cdot 10^{-7}$	$2.263 \cdot 10^{-2}$	$3.297 \cdot 10^{-11}$
$\sigma_{noise} = 100.0$	$2.607 \cdot 10^{-2}$	$6.235 \cdot 10^{-11}$	$4.829 \cdot 10^{-3}$	$3.186 \cdot 10^{-14}$	$3.601 \cdot 10^{-3}$	$8.549 \cdot 10^{-15}$

$\sigma_{src} = 0.8$:						
	$\sigma_{fluct} = 10^{-3}$		$\sigma_{fluct} = 10^{-4}$		$\sigma_{fluct} = 10^{-5}$	
	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$	χ^2	$p_{pot.fluct.}$
$\sigma_{noise} = 5.0$	4.921	$1.589 \cdot 10^{-1}$	2.729	$2.593 \cdot 10^{-2}$	$6.205 \cdot 10^{-1}$	$7.658 \cdot 10^{-5}$
$\sigma_{noise} = 12.0$	1.629	$3.928 \cdot 10^{-3}$	$3.039 \cdot 10^{-1}$	$3.509 \cdot 10^{-6}$	$5.405 \cdot 10^{-2}$	$1.639 \cdot 10^{-9}$
$\sigma_{noise} = 100.0$	$1.196 \cdot 10^{-2}$	$1.879 \cdot 10^{-12}$	$1.219 \cdot 10^{-2}$	$2.044 \cdot 10^{-12}$	$1.778 \cdot 10^{-2}$	$1.117 \cdot 10^{-11}$

Table 4.2 – χ^2-values with respect to the noise level and the corresponding probability that the data is a measurement of the potential fluctuations for the power spectra of different sources, noises and fluctuation scales. These values are for an input power spectrum of $P(l) \sim l^{-6}$ and a single measurement of the lens system (blue errorbars).

For a signal to noise ratio smaller than unity ($\sigma_{noise} = 100.0$), every measurement is insignificant with respect to noise. This is equivalent to what was seen in the plots of the results. The only powerspectra that can be extracted from the noise with a reasonable certainty are the ones for the two largest sources with a noise level of 5.0 intesity units and a potential fluctation power spectrum with a slope of -4. The probability that we would measure the fluctuations under those conditions are 96.74% and 99.43% for the lowest noise level with $\sigma_{src} = 0.5$ and 0.8 respectively. These noise levels correspond to a mean S/N ratio over the image of $\sim$ 7-9. A steeper fluctuation spectrum decreases the probability of measurement, because here the smallest and intermediate scales are cut off by the steepness of the spectrum, resulting in the flattening to the noise level seen in the figures.
The only way here to increase the accuracy of measurement is to observe more than one gravitational lens event. In the case of one hundred observations, significant measurement will be possible for many more situations. However, as was mentioned before in this discussion, it will be problematic to find a hundred lens systems with almost the same properties.
One thing that needs to be taken into account is that the χ^2 statistic is determined for the entire power spectrum, so all the ten datapoints that were calculated. Even though the data at large k-values will mostly be dominated by the noise, it should still be possible to measure the low k part of the spectrum as long as the errorbars don't extend below the noise power spectrum.

Chapter 5

Conclusion

The goal of this thesis was to determine if it will be possible under certain conditions to measure the effect of potential fluctuations on the image of a gravitationally lensed source. We simulated the lensing of several different source galaxies by an elliptical lens galaxy. The lens potential was then disturbed by the addition of a Gaussian random field which resembles density fluctuations in that galaxy. Afterwards we artificially added random noise to the image and subtracted a smooth model from it to obtain residuals that contain both the noise and the result from lensing the potential fluctuations. A power spectrum was then generated from the residuals which gives the amplitude with which certain scales are present in the residuals image.

The different parameters that we varied in order to constrain observations of the residuals power spectrum were:

- The noise level: We added noise to the observations with three maximum levels (5.0, 12.0 and 100.0 intensity units). The highest noise level gives a signal to noise ratio smaller than unity on the entire image.
- The scale of the potential fluctuations: Variances of the fluctuations for three different orders of magnitude were implemented. The values that we used were $\sigma^2_{fluct} = 10^{-3}, 10^{-4}$ and 10^{-5}.
- The size of the source object: An exponential function was used to model the source galaxy with a peak brightness of 100 intesity units and widths $\sigma_{src} = 0.1, 0.25, 0.5$ and 0.8.
- The slope of the power spectrum of the potential flucatations: We modeled fluctuations with power spectra with slopes of -4 and -6. The steeper spectrum gives smoother fluctuations that do not contain many small scales.

From the results we find that noise starts dominating the power spectra at the smallest scales and that especially the smallest fluctuations will be difficult to measure accurately. Larger source objects cause the amplitude of the spectra to increase slightly, making measurements of the lens potential fluctuations better and there does not appear to be a significant change in the shape of the power spectra. A steeper potential fluctuation power spectrum results in a cut-off of the smaller scales, which will therefore also result in a reduction of the power at the smaller scales in the residuals. Furthermore, a χ^2 analysis showed that the accuracy for extracting the power spectrum due to potential fluctuations from the noise will be problematic. Only the largest sources combined with the most intens potential fluctuations and a low noise level result in reasonable probabilities for a deviation from the noise power spectrum. This does not mean that we will not be able to measure some part of the power spectrum at all, as for the lower noise levels only smallest scales are completely dominated by noise. Another option is to measure more than one lens event and combining the results. A hundred observations of the same type of lens system can increase the accuracy of the measurement, but in general not all real lens systems will have the same composition.

5.1 Future Research

There is still a lot of work that can be done to expand upon the research performed for this thesis. This section gives an overview of some follow-up options.

We only looked at the general characteristics of the power spectra, but a fit of the curve was not made. One could for example try to see if the power spectra follow a powerlaw and include a constant noise component. This could then be compared to the input spectrum of the potential fluctuations, to see if it would be possible to immediately recognize what kind of power spectrum the fluctuations in the lens follow. Furthermore, a more detailed statistical analysis could be performed to get a clearer view of which parts of the spectrum can be measured significantly. Some of the data at the larger scales does rise above the noise level and therefore one would expect those scales to get a higher significance in that measument.

An analytical equation for the power spectrum of the residuals due to the addition of potential fluctuations to the lens potential was derived in the bachelor thesis by Bus (2012)[20]. To solve it analytically would be very complicated however. One problem with the relation is that it is only valid in single images of the source, whereas our simulation models an entire lensing event which consists of multiple images. Therefore at the lowest k-range, there will be a correlation between points in separate images and the equation breaks down. So in order to compare their result with the ones from our simulations, one would have to look at the higher k part of the power spectra. In some of the plots there is a very small bump present around $\frac{k}{2\pi} = 0.8 - 0.9 arcsec^{-1}$ (for instance in figure 4.2). From our research we cannot make any conclusion about the cause of this increase in power. Work is underway nonetheless to solve the equations numerically and therefore future work might reveal more.

One thing that can help observations is the resolution of the telescope. In our simulation we used parameters for the Hubble Space Telescope. One could repeat the simulations for Keck Adaptive Optics (higher resolution, but also more noise) or the future EUCLID telescope (lower resolution, but also less noise) and see which of the two would be favourable for observations of the residuals power spectrum. These results could then be used to determine the optimum observing strategy. To expand upon this, it is also useful to simulate more than one lens system. Our model is typical lens system, but for real observations we expect to find other geometries of the lens system which could also affect the results. This can then be used to simulate a sample of real lenses and apply the same measuments to that to constrain the differences arising from other geometries. The next step would then be to measure power spectra from real data and using Monte Carlo statistics it might be possible to constrain which fluctuation scales are present in real lenses.

5.2 Acknowledgements

There are a number of people that were essential in making this project possible. Most of all I am grateful to my supervisor Leon Koopmans. I am glad it was possible to do this project under his guidance. He was very patient when we ran into problems with the code and always took the time to give advice or explaining steps in detail. Another person that I would like thank is Sander Bus, for first coming up with the idea for this research as a follow-up of his own. We had a lot of very useful discussions and he helped me to easily get more acquinted to the theory of gravitational lensing. Furthermore I am grateful for the help with this project provided by Patrick Bos, Boudewijn Hut, Job Feldbrugge, Ronniy Joseph, Daniel Siepman, Marlies Spijkman and Rien van de Weygaert. Last, but not least I would like to thank Omar Choudhurry for taking the time to help with tracing down some problems in the code and to give suggestions for improving this thesis.

Bibliography

[1] P. Schneider. Gravitational lensing as a probe of structure. *ArXiv Astrophysics e-prints*, June 2003.

[2] Galaxy cluster abell 2218's gravitational lens, . URL `http://hubblesite.org/gallery/album/entire/pr2001032b/large_web/`.

[3] D. Valls-Gabaud. The conceptual origins of gravitational lensing. In J.-M. Alimi and A. Füzfa, editors, *Albert Einstein Century International Conference*, volume 861 of *American Institute of Physics Conference Series*, pages 1163–1163, November 2006. doi: 10.1063/1.2399715.

[4] R. Narayan and M. Bartelmann. Lectures on Gravitational Lensing. *ArXiv Astrophysics e-prints*, June 1996.

[5] The double QSO 0957+561, . URL `http://www.astr.ua.edu/keel/agn/q0957.gif`.

[6] C.S. Kochanek, E.E. Falco, C. Impey, J. Lehar, B. McLeod and H.-W. Rix. CASTLES Survey. URL `http://www.cfa.harvard.edu/castles/`.

[7] W. Zheng, M. Postman, A. Zitrin, J. Moustakas, X. Shu, S. Jouvel, O. Host, A. Molino, L. Bradley, D. Coe, L. A. Moustakas, M. Carrasco, H. Ford, N. Benıtez, T. R. Lauer, S. Seitz, R. Bouwens, A. Koekemoer, E. Medezinski, M. Bartelmann, T. Broadhurst, M. Donahue, C. Grillo, L. Infante, S. Jha, D. D. Kelson, O. Lahav, D. Lemze, P. Melchior, M. Meneghetti, J. Merten, M. Nonino, S. Ogaz, P. Rosati, K. Umetsu, and A. van der Wel. A highly magnified candidate for a young galaxy seen when the Universe was 500 Myrs old. *ArXiv e-prints*, April 2012.

[8] T. Treu. Strong Lensing by Galaxies. ARA&A, 48:87–125, September 2010. doi: 10.1146/annurev-astro-081309-130924.

[9] Ronald J. Tallarida. *Pocket Book of Integrals and Mathematical Formulas.* Taylor & Francis Group, 4th edition, 2008.

[10] Rien van de Weygaert. Large Scale Structure lecture notes.

[11] L. V. E. Koopmans. Gravitational imaging of cold dark matter substructures. MNRAS, 363: 1136–1144, November 2005. doi: 10.1111/j.1365-2966.2005.09523.x.

[12] R. Kormann, P. Schneider, and M. Bartelmann. Isothermal elliptical gravitational lens models. A&A, 284:285–299, April 1994.

[13] Discrete Fourier Transform (numpy.fft), . URL `http://docs.scipy.org/doc/numpy/reference/routines.fft.html`.

[14] Rien van de Weygaert. Large Scale Structure 2009: Tutorial IV: (Gaussian) Random Fields, 2009.

[15] Numpy.fft.fftfreq, . URL `http://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fftfreq.html#numpy.fft.fftfreq`.

[16] G. Marsaglia and T. A. Bray. A convenient method for generating normal variables. *SIAM Review*, 6(3):pp. 260–264, 1964. ISSN 00361445. URL `http://www.jstor.org/stable/2027592`.

[17] Marc A. Murison. A Method for Directly Generating a Gaussian Distribution with Nonunit Variance and Nonzero Mean from Uniform Random Deviates, 2000.

[18] M. van der Klis. Fourier techniques in X-ray timing. In H. Ögelman and E. P. J. van den Heuvel, editors, *Timing Neutron Stars*, page 27, 1989.

[19] OpenOffice Calc CHIDIS function, .

[20] Sander Bus. Power spectrum analysis of galaxy potentials using strong lensing events. Bachelor thesis, University of Groningen, 2012.

Appendix A

Plots For Intermediate Source Sizes

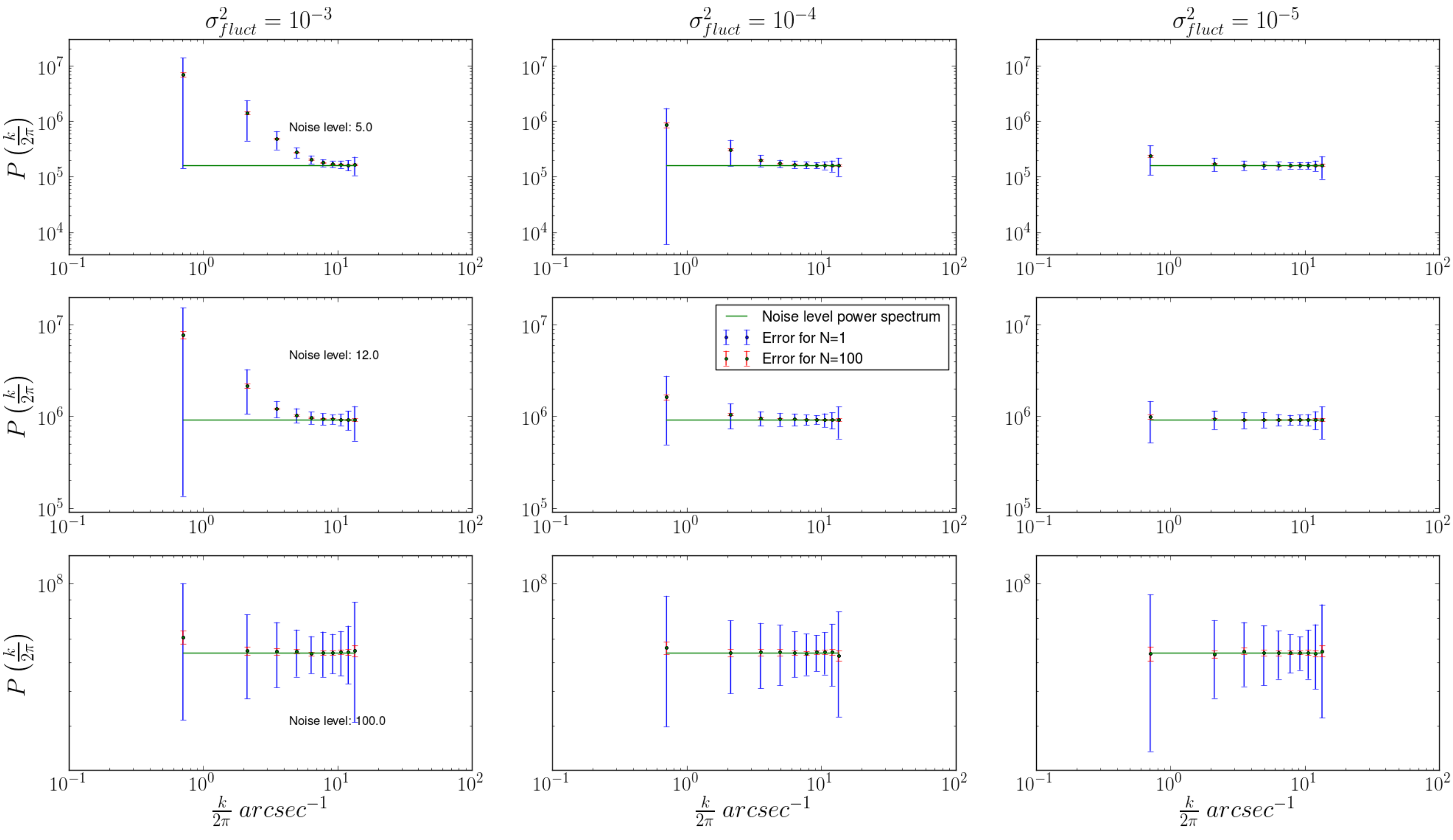


Figure A.1 – Power spectra with a source size $\sigma_{src} = 0.25$. The rows represent different noise levels, from top to bottom 5.0, 12.0 and 100.0 respectively. The horizontal green line gives the power spectrum expected for that noise level. Every column has the same fluctuation sizes, from left to right $\sigma^2_{flucts} = 10^{-3}, 10^{-4}, 10^{-5}$ respectively. Blue errorbars give the error for a single measurement, whereas the red errorbars would be the error after 100 observations. The fluctuation inputspectrum was $P(l) \propto l^{-4}$.

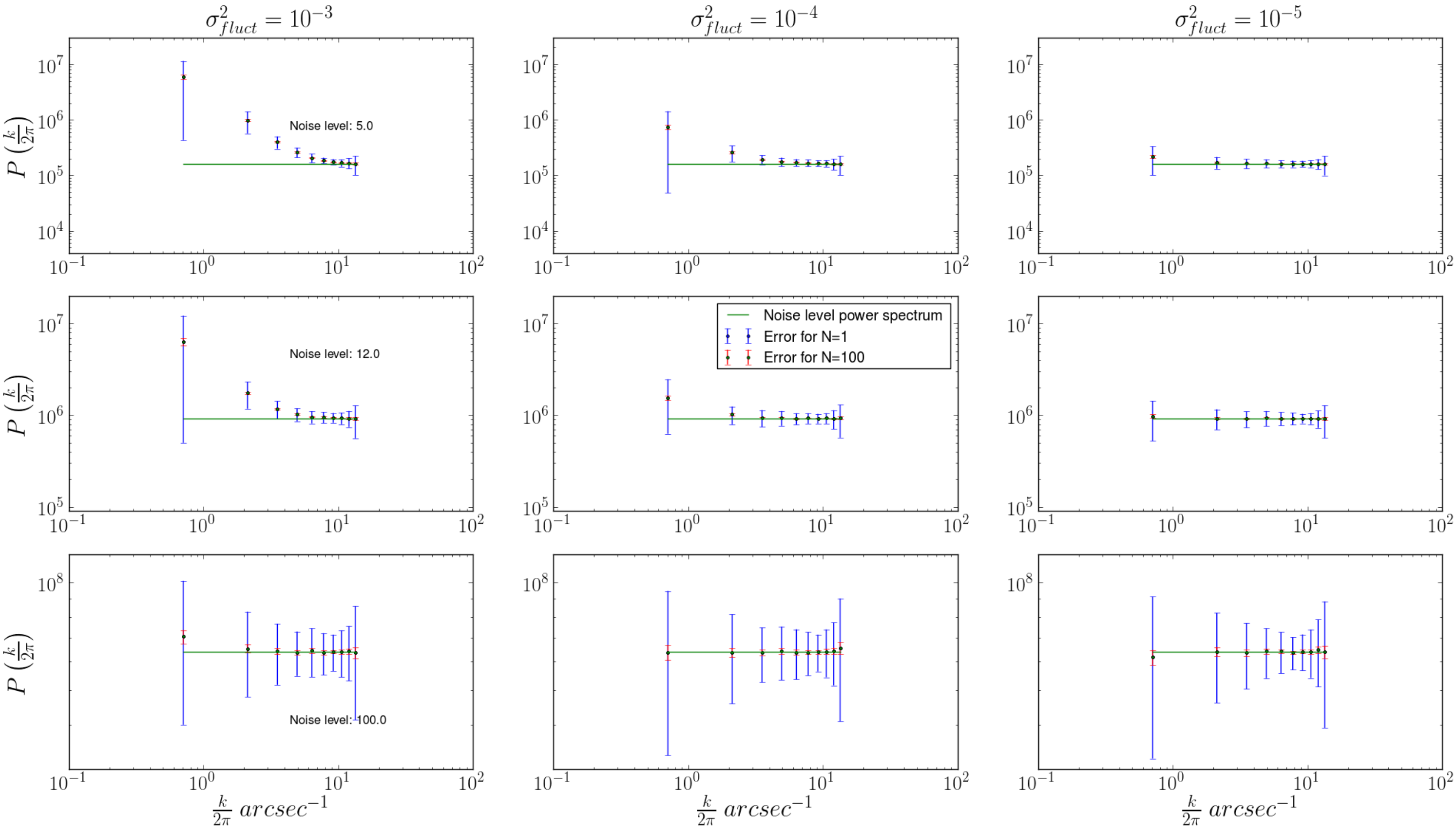


Figure A.2 – Power spectra with a source size $\sigma_{src} = 0.5$. The rows represent different noise levels, from top to bottom 5.0, 12.0 and 100.0 respectively. The horizontal green line gives the power spectrum expected for that noise level. Every column has the same fluctuation sizes, from left to right $\sigma^2_{flucts} = 10^{-3}, 10^{-4}, 10^{-5}$ respectively. Blue errorbars give the error for a single measurement, whereas the red errorbars would be the error after 100 observations. The fluctuation inputspectrum was $P(l) \propto l^{-4}$.

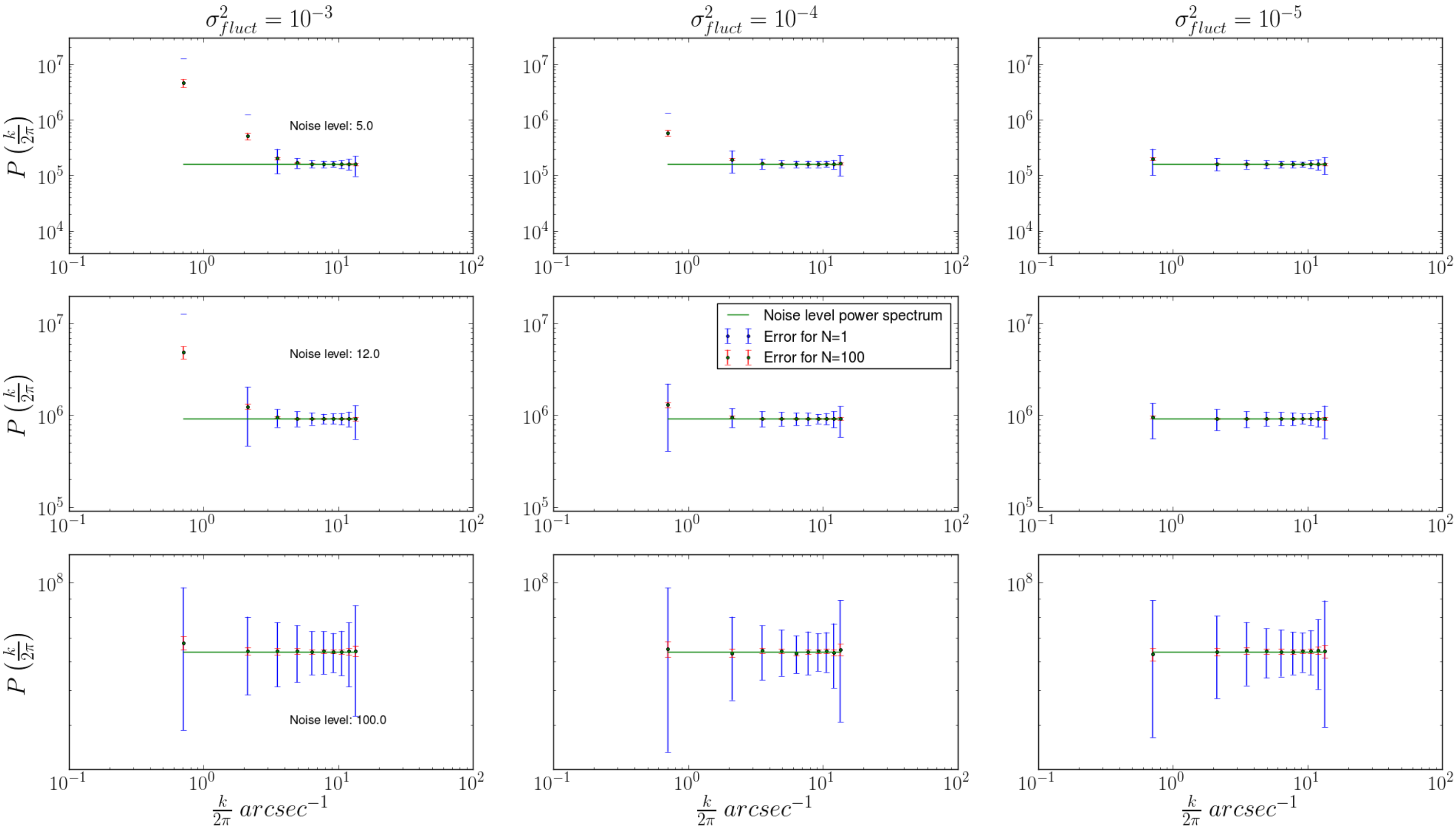


Figure A.3 – Power spectra with a source size $\sigma_{src} = 0.25$. The rows represent different noise levels, from top to bottom 5.0, 12.0 and 100.0 respectively. The horizontal green line gives the power spectrum expected for that noise level. Every column has the same fluctuation sizes, from left to right $\sigma^2_{flucts} = 10^{-3}, 10^{-4}, 10^{-5}$ respectively. Blue errorbars give the error for a single measurement, whereas the red errorbars would be the error after 100 observations. The fluctuation inputspectrum was $P(l) \propto l^{-6}$.

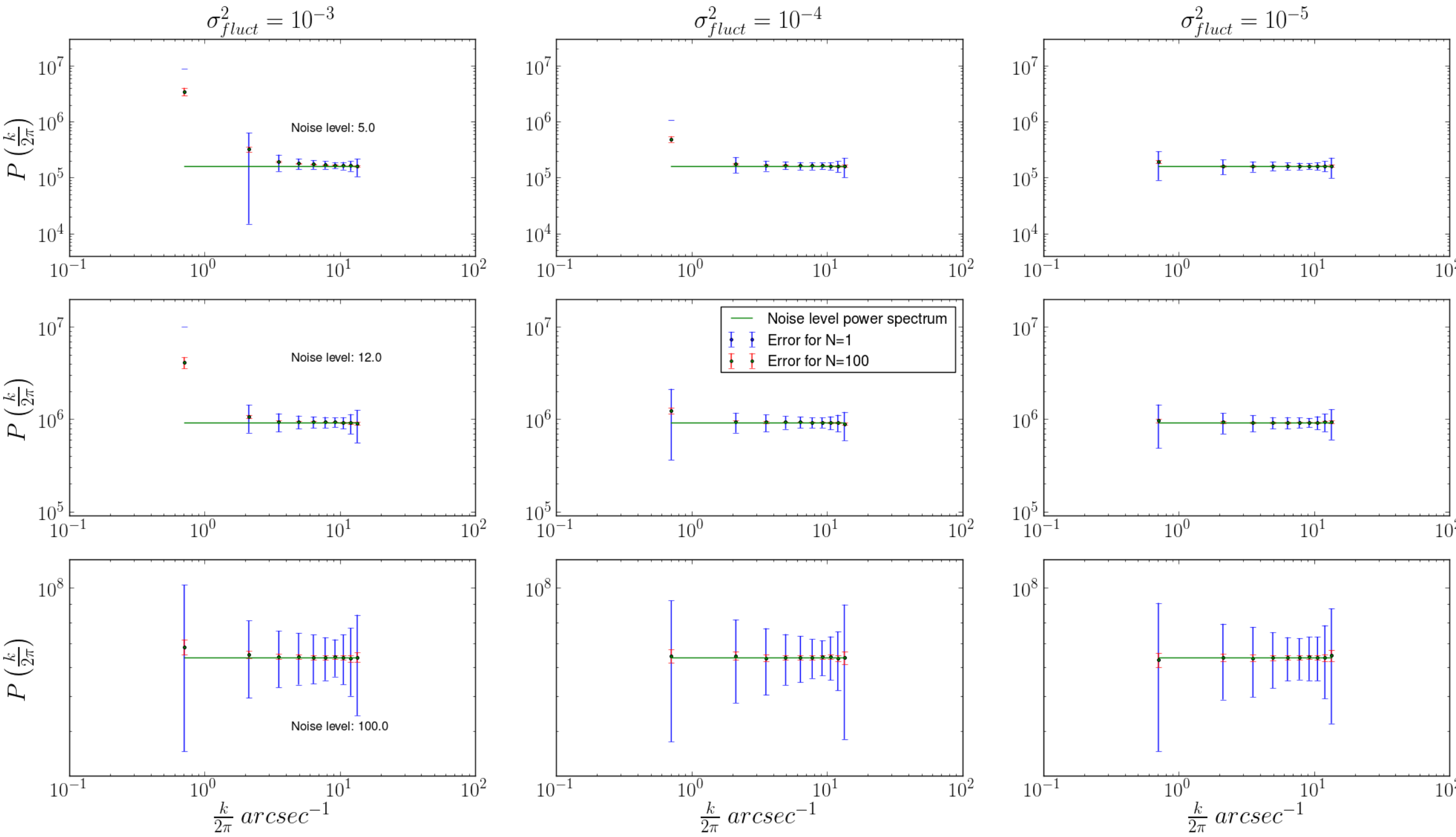


Figure A.4 – Power spectra with a source size $\sigma_{src} = 0.5$. The rows represent different noise levels, from top to bottom 5.0, 12.0 and 100.0 respectively. The horizontal green line gives the power spectrum expected for that noise level. Every column has the same fluctuation sizes, from left to right $\sigma^2_{flucts} = 10^{-3}, 10^{-4}, 10^{-5}$ respectively. Blue errorbars give the error for a single measurement, whereas the red errorbars would be the error after 100 observations. The fluctuation inputspectrum was $P(l) \propto l^{-6}$.

Appendix B

Lensing Code

```
#! /usr/bin/env python

# ###########################################################################################
# Loading packages
# ###########################################################################################

#Import standard packages

import sys
import pyfits
from numpy.oldnumeric import *
import numpy.numarray.random_array as ranlib
import math as m

#Include ‘personal’ packages

sys.path = sys.path + ['/net/dataserver1/data/student/kooistra/
    onderzoek/sparse/lib64/python/']

from pysparse import spmatrix
from gaussran2 import * #importing gaussian random field code
from residuals import * #importing code for power spectrum creation of
    residuals
from histogram import * #importing code for creating a distribution of
    the random fields


# #########################################################################################
# Deflection angle for SIE lens + External Shear at position (x,y)
# #########################################################################################
```

```python

def deflect_SIE(lens, x, y):

    # SIE lens model

    tr = pi*(lens.th/180.0)+pi/2.0

    sx = x-lens.x0
    sy = y-lens.y0

    cs = cos(tr)
    sn = sin(tr)

    sx_r =  sx*cs+sy*sn
    sy_r = -sx*sn+sy*cs

    psi = sqrt(lens.fl**2.0 * (lens.rc**2.0 + sx_r**2.0) + sy_r**2.0)

    dx_tmp = (lens.bl*sqrt(lens.fl)/sqrt(1.0-lens.fl**2.0))* arctan(sqrt(1.0-lens.fl**2.0)*sx_r/(psi+lens.rc))
    dy_tmp = (lens.bl*sqrt(lens.fl)/sqrt(1.0-lens.fl**2.0))*arctanh(sqrt(1.0-lens.fl**2.0)*sy_r/(psi+lens.rc*lens.fl**2.0))

    dx = dx_tmp*cs - dy_tmp*sn
    dy = dx_tmp*sn + dy_tmp*cs

    # external shear

    tr2 = pi*(lens.sa/180.0)
    cs2 = cos(2.0*tr2)
    sn2 = sin(2.0*tr2)

    dx2 = lens.ss*(cs2*sx+sn2*sy)
    dy2 = lens.ss*(sn2*sx-cs2*sy)

    return array([dx+dx2, dy+dy2])

# ################################################################################
# Convergence for SIE + external shear
# ################################################################################

def convergence_SIE(lens, x, y):

    # SIE lens model

    tr = pi*(lens.th/180.0)+pi/2.0

    sx = x-lens.x0
    sy = y-lens.y0

```

```
    cs = cos(tr)
    sn = sin(tr)

    sx_r =  sx*cs+sy*sn
    sy_r = -sx*sn+sy*cs

    psi = sqrt(lens.fl**2.0 * (lens.rc**2.0 + sx_r**2.0) + sy_r**2.0)

    kappa_tmp = (0.5*lens.bl*sqrt(lens.fl)/psi)

    return kappa_tmp

# ##############################################################################
# Potential for SIE + external shear
# ##############################################################################

def potential(lens, x, y):

    # SIE lens model

    tr = pi*(lens.th/180.0)+pi/2.0

    sx = x-lens.x0
    sy = y-lens.y0

    cs = cos(tr)
    sn = sin(tr)

    sx_r =  sx*cs+sy*sn
    sy_r = -sx*sn+sy*cs

    psi = sqrt(lens.fl**2.0 * (lens.rc**2.0 + sx_r**2.0) + sy_r**2.0)

    dx_tmp = (lens.bl*sqrt(lens.fl)/sqrt(1.0-lens.fl**2.0))* arctan(
        sqrt(1.0-lens.fl**2.0)*sx_r/(psi+lens.rc))
    dy_tmp = (lens.bl*sqrt(lens.fl)/sqrt(1.0-lens.fl**2.0))*arctanh(
        sqrt(1.0-lens.fl**2.0)*sy_r/(psi+lens.rc*lens.fl**2.0))

    pot_SIE = sx_r*dx_tmp + sy_r*dy_tmp - 0.5*lens.bl*sqrt(lens.fl)*
        lens.rc*log((psi+lens.rc)**2.0+(1.0-(lens.fl**2.0))*(sx_r**2.0)
        )

    # external shear

    tr2 = pi*(lens.sa/180.0)
    cs2 = cos(2.0*tr2)
    sn2 = sin(2.0*tr2)

    pot_exts = lens.ss*(sn2*sx*sy + 0.5*cs2*(sx**2.0-sy**2.0))

```

```python
    return pot_SIE + pot_exts

# ############################################################################
# Convergence from potential correction
# ############################################################################

def convergence(gdat,ldat1,ldat2):

    # poisson equation

    gpot_dx = (gdat.gpot.xmax - gdat.gpot.xmin)/(gdat.gpot.dim1-1)
    gpot_dy = (gdat.gpot.ymax - gdat.gpot.ymin)/(gdat.gpot.dim2-1)

    kappa      = zeros(gdat.gpot.dim1*gdat.gpot.dim2,'d')
    kappa_mask = zeros(gdat.gpot.dim1*gdat.gpot.dim2,'d')

    for i in range(0,gdat.gpot.dim1):
        for j in range(0,gdat.gpot.dim2):

            x = gdat.gpot.xmin + i*gpot_dx
            y = gdat.gpot.ymin + j*gpot_dy

            kappa[i+j*gdat.gpot.dim1] = convergence_SIE(ldat2,x,y)

    return kappa

# ############################################################################
# Total (non-corrected) potential grid
# ############################################################################

def pot_grid(gdat,lens1,lens2,gen,src_sig,noise,sigpow,pspec):

    gpot_dx = (gdat.gpot.xmax-gdat.gpot.xmin)/(gdat.gpot.dim1-1.0)
    gpot_dy = (gdat.gpot.ymax-gdat.gpot.ymin)/(gdat.gpot.dim2-1.0)

    pot_nc = zeros(gdat.gpot.dim1*gdat.gpot.dim2,'d')

    for i in range(gdat.gpot.dim1):
        for j in range(gdat.gpot.dim2):
            xx = i*gpot_dx + gdat.gpot.xmin
            yy = j*gpot_dy + gdat.gpot.ymin
            pot_nc[i+j*gdat.gpot.dim1] = potential(lens1,xx,yy)+potential(lens2,xx,yy)

    pot_nc = pot_nc + n.reshape(substruct(gdat.gpot,gen,gdat.gpot.fluctsig,src_sig,noise,sigpow,pspec),n.shape(pot_nc)) # adding
```

```
        potential fluctuations

    return pot_nc

#
 ####################################################################################
# Correct t,u for small numerical deviations from either 0.0 or 1.0
#
 ####################################################################################

def corr_x_i(x,i):

    x2 = x
    i2 = i

    if (abs(x)<=1.0e-8):
        if (x<0.0):
            i2=i+1
        x2 = 0.0

    if (abs(x-1.0)<=1.0e-8):
        if (x<1.0):
            i2=i+1
        x2 = 1.0

    return x2,i2

#
 ####################################################################################
# Deflection angle for linear-correction grid
#
 ####################################################################################

def deflect_grid(gdat, xx, yy):

    gpot_dx = (gdat.gpot.xmax - gdat.gpot.xmin)/(gdat.gpot.dim1-1)
    gpot_dy = (gdat.gpot.ymax - gdat.gpot.ymin)/(gdat.gpot.dim2-1)

    i1 = int(floor((xx-gdat.gpot.xmin)/gpot_dx))
    j1 = int(floor((yy-gdat.gpot.ymin)/gpot_dy))

    t = (xx - (i1*gpot_dx+gdat.gpot.xmin))/gpot_dx
    u = (yy - (j1*gpot_dy+gdat.gpot.ymin))/gpot_dy

    # snap to nearest pixel if very close

    cxi = corr_x_i(t,i1)
    t  = cxi[0]
    i1 = cxi[1]

```

```python
    cxi = corr_x_i(u,j1)
    u  = cxi[0]
    j1 = cxi[1]

    # if pixel is inside gpot grid, then continue

    dgdx = 0.0
    dgdy = 0.0

    if (i1 in range(1,gdat.gpot.dim1-2) and j1 in range(1,gdat.gpot.dim2-2)):

        # determine fraction of pixel

        t = (xx - (i1*gpot_dx+gdat.gpot.xmin))/gpot_dx
        u = (yy - (j1*gpot_dy+gdat.gpot.ymin))/gpot_dy

        # dpot on grid points enclosing the pixel (xx,yy)

        dy1dx = (gdat.gpot.data[i1+1+j1*gdat.gpot.dim1]-gdat.gpot.data[i1-1+j1*gdat.gpot.dim1])/(2.0*gpot_dx)
        dy2dx = (gdat.gpot.data[i1+2+j1*gdat.gpot.dim1]-gdat.gpot.data[i1+j1*gdat.gpot.dim1])/(2.0*gpot_dx)
        dy3dx = (gdat.gpot.data[i1+2+(j1+1)*gdat.gpot.dim1]-gdat.gpot.data[i1+(j1+1)*gdat.gpot.dim1])/(2.0*gpot_dx)
        dy4dx = (gdat.gpot.data[i1+1+(j1+1)*gdat.gpot.dim1]-gdat.gpot.data[i1-1+(j1+1)*gdat.gpot.dim1])/(2.0*gpot_dx)

        dy1dy = (gdat.gpot.data[i1+(j1+1)*gdat.gpot.dim1]-gdat.gpot.data[i1+(j1-1)*gdat.gpot.dim1])/(2.0*gpot_dy)
        dy2dy = (gdat.gpot.data[i1+1+(j1+1)*gdat.gpot.dim1]-gdat.gpot.data[i1+1+(j1-1)*gdat.gpot.dim1])/(2.0*gpot_dy)
        dy3dy = (gdat.gpot.data[i1+1+(j1+2)*gdat.gpot.dim1]-gdat.gpot.data[i1+1+j1*gdat.gpot.dim1])/(2.0*gpot_dy)
        dy4dy = (gdat.gpot.data[i1+(j1+2)*gdat.gpot.dim1]-gdat.gpot.data[i1+j1*gdat.gpot.dim1])/(2.0*gpot_dy)

        dgdx = (1.0-t)*(1.0-u)*dy1dx + t*(1.0-u)*dy2dx + t*u*dy3dx + (1.0-t)*u*dy4dx
        dgdy = (1.0-t)*(1.0-u)*dy1dy + t*(1.0-u)*dy2dy + t*u*dy3dy + (1.0-t)*u*dy4dy

    return array([dgdx,dgdy])

# ##############################################################################
# Deflection angle for last change in linear-correction grid
# ##############################################################################

def deflect_dpot(gdat, dpot, xx, yy):

```

```
    gpot_dx = (gdat.gpot.xmax - gdat.gpot.xmin)/(gdat.gpot.dim1-1)
    gpot_dy = (gdat.gpot.ymax - gdat.gpot.ymin)/(gdat.gpot.dim2-1)

    i1 = int(floor((xx-gdat.gpot.xmin)/gpot_dx))
    j1 = int(floor((yy-gdat.gpot.ymin)/gpot_dy))

    t = (xx - (i1*gpot_dx+gdat.gpot.xmin))/gpot_dx
    u = yy - (j1*gpot_dy+gdat.gpot.ymin)/gpot_dy

    # snap to nearest pixel if very close

    cxi = corr_x_i(t,i1)
    t  = cxi[0]
    i1 = cxi[1]

    cxi = corr_x_i(u,j1)
    u  = cxi[0]
    j1 = cxi[1]

    # if pixel is inside gpot grid, then continue

    dgdx = 0.0
    dgdy = 0.0

    if (i1 in range(1,gdat.gpot.dim1-2) and j1 in range(1,gdat.gpot.dim2-2)):

        # dpot on grid points enclosing the pixel (xx,yy)

        dy1dx = (dpot[i1+1+j1*gdat.gpot.dim1]-dpot[i1-1+j1*gdat.gpot.dim1])/(2.0*gpot_dx)
        dy2dx = (dpot[i1+2+j1*gdat.gpot.dim1]-dpot[i1+j1*gdat.gpot.dim1])/(2.0*gpot_dx)
        dy3dx = (dpot[i1+2+(j1+1)*gdat.gpot.dim1]-dpot[i1+(j1+1)*gdat.gpot.dim1])/(2.0*gpot_dx)
        dy4dx = (dpot[i1+1+(j1+1)*gdat.gpot.dim1]-dpot[i1-1+(j1+1)*gdat.gpot.dim1])/(2.0*gpot_dx)

        dy1dy = (dpot[i1+(j1+1)*gdat.gpot.dim1]-dpot[i1+(j1-1)*gdat.gpot.dim1])/(2.0*gpot_dy)
        dy2dy = (dpot[i1+1+(j1+1)*gdat.gpot.dim1]-dpot[i1+1+(j1-1)*gdat.gpot.dim1])/(2.0*gpot_dy)
        dy3dy = (dpot[i1+1+(j1+2)*gdat.gpot.dim1]-dpot[i1+1+j1*gdat.gpot.dim1])/(2.0*gpot_dy)
        dy4dy = (dpot[i1+(j1+2)*gdat.gpot.dim1]-dpot[i1+j1*gdat.gpot.dim1])/(2.0*gpot_dy)

        dgdx = (1.0-t)*(1.0-u)*dy1dx + t*(1.0-u)*dy2dx + t*u*dy3dx + (1.0-t)*u*dy4dx
        dgdy = (1.0-t)*(1.0-u)*dy1dy + t*(1.0-u)*dy2dy + t*u*dy3dy + (1.0-t)*u*dy4dy

    return array([dgdx,dgdy])

```

```python

# ##########################################################################
# Deflection matrix data
# ##########################################################################

def deflect_info_grid(gdat, xx, yy):

    gpot_dx = (gdat.gpot.xmax - gdat.gpot.xmin)/(gdat.gpot.dim1-1)
    gpot_dy = (gdat.gpot.ymax - gdat.gpot.ymin)/(gdat.gpot.dim2-1)

    i1 = int(floor((xx-gdat.gpot.xmin)/gpot_dx))
    j1 = int(floor((yy-gdat.gpot.ymin)/gpot_dy))

    t = (xx - (i1*gpot_dx+gdat.gpot.xmin))/gpot_dx
    u = (yy - (j1*gpot_dy+gdat.gpot.ymin))/gpot_dy

    cxi = corr_x_i(t,i1)
    t  = cxi[0]
    i1 = cxi[1]

    cxi = corr_x_i(u,j1)
    u  = cxi[0]
    j1 = cxi[1]

    if (i1 in range(1,gdat.gpot.dim1-2) and j1 in range(1,gdat.gpot.dim2-2)):

        # determine fraction of pixel

        return array([-(1.0-t)*(1.0-u), -(1.0-t)*u, -t*(1.0-u), -t*u, (1.0-t)*(1.0-u), (1.0-t)*u, t*(1.0-u), t*u])/(2.0*gpot_dx), \
               array([-(1.0-t)*(1.0-u), -(1.0-t)*u, (1.0-t)*(1.0-u), (1.0-t)*u, -t*(1.0-u), -t*u, t*(1.0-u), t*u])/(2.0*gpot_dy), \
               array([i1,j1])

    return array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]),array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]),array([i1,j1])

# ##########################################################################
# Source position
# ##########################################################################

def source_pos(ldat1,ldat2,gdat,x,y,fl_dpot):

```

```
    if (fl_dpot == 1):
        return array([x,y])-(deflect_SIE(ldat1,x,y)+deflect_SIE(ldat2,
            x,y)+deflect_grid(gdat,x,y))
    else:
        return array([x,y])-(deflect_SIE(ldat1,x,y)+deflect_SIE(ldat2,
            x,y))

#
    ################################################################################
# Operator to determine source and potential **** simultaneous ****
#
    ################################################################################

def source_psi_op(ldat1,ldat2,gdat,data_mask,BO,fl_dpot,fl_corr):

    img_dx = (gdat.img.xmax-gdat.img.xmin)/(gdat.img.dim1-1.0)
    img_dy = (gdat.img.ymax-gdat.img.ymin)/(gdat.img.dim2-1.0)

    src_dx = (gdat.src.xmax-gdat.src.xmin)/(gdat.src.dim1-1.0)
    src_dy = (gdat.src.ymax-gdat.src.ymin)/(gdat.src.dim2-1.0)

    gpot_dx = (gdat.gpot.xmax - gdat.gpot.xmin)/(gdat.gpot.dim1-1.0)
    gpot_dy = (gdat.gpot.ymax - gdat.gpot.ymin)/(gdat.gpot.dim2-1.0)

    SPO = spmatrix.ll_mat(gdat.img.dim1*gdat.img.dim2, gdat.src.dim1*
        gdat.src.dim2 + gdat.gpot.dim1*gdat.gpot.dim2)

    lmask = zeros(gdat.img.dim1*gdat.img.dim2,'d')
    smask = zeros(gdat.src.dim1*gdat.src.dim2,'d')
    pmask = zeros(gdat.gpot.dim1*gdat.gpot.dim2,'d')

    ##########################################
    # Fill in the lens_operator part
    ##########################################

    for i in range(gdat.img.dim1):
        for j in range(gdat.img.dim2):

            # (i,j) corresponds to physical scale (xx,yy)
            #  i -> row
            #  j -> col

            xx = i*img_dx + gdat.img.xmin
            yy = j*img_dy + gdat.img.ymin

            # get physical position in source plane

            sv = source_pos(ldat1,ldat2,gdat,xx,yy,fl_dpot)

            # corresponding pixel in source plane

            i1 = int(floor((sv[0]-gdat.src.xmin)/src_dx))
```

```
j1 = int(floor((sv[1]-gdat.src.ymin)/src_dy))

# if pixel is inside source-plane grid, then continue

t = (sv[0] - (i1*src_dx+gdat.src.xmin))/src_dx
u = (sv[1] - (j1*src_dy+gdat.src.ymin))/src_dy

cxi = corr_x_i(t,i1)
t  = cxi[0]
i1 = cxi[1]

cxi = corr_x_i(u,j1)
u  = cxi[0]
j1 = cxi[1]

if (i1 in range(gdat.src.dim1-1) and
    j1 in range(gdat.src.dim2-1)):

    # determine fraction of pixel

    # This is an array, not a grid, so indexing is different

    SPO[i+j*gdat.img.dim1, i1+j1*gdat.src.dim1]         = (1.0-t)*(1.0-u)
    SPO[i+j*gdat.img.dim1, i1+1+j1*gdat.src.dim1]       = t*(1.0-u)
    SPO[i+j*gdat.img.dim1, i1+1+(j1+1)*gdat.src.dim1]   = t*u
    SPO[i+j*gdat.img.dim1, i1+(j1+1)*gdat.src.dim1]     = (1.0-t)*u

    lmask[i+j*gdat.img.dim1]              = 1.0

    if (data_mask[i+j*gdat.img.dim1] == 1.0):

        smask[i1+j1*gdat.src.dim1]         = 1.0
        smask[i1+1+j1*gdat.src.dim1]       = 1.0
        smask[i1+1+(j1+1)*gdat.src.dim1]   = 1.0
        smask[i1+(j1+1)*gdat.src.dim1]     = 1.0

else:

    if ((i1 == (gdat.src.dim1-1)) and (j1 in range(gdat.src.dim2-2)) and (t == 0.0)):

        lmask[i+j*gdat.img.dim1]              = 1.0
        SPO[i+j*gdat.img.dim1, i1+j1*gdat.src.dim1]         = (1.0-u)
        SPO[i+j*gdat.img.dim1, i1+(j1+1)*gdat.src.dim1]     = u

        if (data_mask[i+j*gdat.img.dim1] == 1.0):
            smask[i1+j1*gdat.src.dim1]         = 1.0
```

```
                        smask[i1+(j1+1)*gdat.src.dim1]    = 1.0

                if ((j1 == (gdat.src.dim2-1)) and (i1 in range(gdat.src.dim1-2)) and (u == 0.0)):

                    lmask[i+j*gdat.img.dim1]          = 1.0
                    SPO[i+j*gdat.img.dim1, i1+j1*gdat.src.dim1] = (1.0-t)
                    SPO[i+j*gdat.img.dim1, i1+1+j1*gdat.src.dim1] = t

                    if (data_mask[i+j*gdat.img.dim1] == 1.0):
                        smask[i1+j1*gdat.src.dim1]        = 1.0
                        smask[i1+1+j1*gdat.src.dim1]      = 1.0

                if ((j1 == (gdat.src.dim2-1)) and (i1 == (gdat.src.dim1-1))\
                    and (u == 0.0) and (t == 0.0)):

                    lmask[i+j*gdat.img.dim1]          = 1.0
                    SPO[i+j*gdat.img.dim1, i1+j1*gdat.src.dim1] = 1.0

                    if (data_mask[i+j*gdat.img.dim1] == 1.0):
                        smask[i1+j1*gdat.src.dim1]        = 1.0


    if (fl_dpot == 1 and fl_corr == 1):

        ###########################################
        # Fill in the linear correction part
        ###########################################

        ########## First determine the source derivative matrix ##########

        SO = spmatrix.ll_mat(gdat.img.dim1*gdat.img.dim2,2*gdat.img.dim1*gdat.img.dim2)

        dd1 = zeros(gdat.src.dim1*gdat.src.dim2,'d')
        dd2 = zeros(gdat.src.dim1*gdat.src.dim2,'d')

        for i in range(gdat.img.dim1-1):
            for j in range(gdat.img.dim2-1):

                SO[(i+j*gdat.img.dim1),2*(i+j*gdat.img.dim1)]   = 0.0
                SO[(i+j*gdat.img.dim1),2*(i+j*gdat.img.dim1)+1]= 0.0

                # (i,j) corresponds to physical scale (xx,yy)
                #   i -> row
                #   j -> col

                xx = i*img_dx + gdat.img.xmin
                yy = j*img_dy + gdat.img.ymin
```

```
        # get physical position in source plane

        sv = source_pos(ldat1,ldat2,gdat,xx,yy,fl_dpot)

        # corresponding pixel in source plane

        i1 = int(floor((sv[0]-gdat.src.xmin)/src_dx))
        j1 = int(floor((sv[1]-gdat.src.ymin)/src_dy))

        t = (sv[0] - (i1*src_dx+gdat.src.xmin))/src_dx
        u = (sv[1] - (j1*src_dy+gdat.src.ymin))/src_dy

        cxi = corr_x_i(t,i1)
        t  = cxi[0]
        i1 = cxi[1]

        cxi = corr_x_i(u,j1)
        u  = cxi[0]
        j1 = cxi[1]

        # if pixel is inside source-plane grid, then continue

        if (i1 in range(gdat.src.dim1-1) and
            j1 in range(gdat.src.dim2-1)):

            # This is an array, not a grid, so indexing is different

            y1 = gdat.src.data[i1+j1*gdat.src.dim1]
            y2 = gdat.src.data[i1+1+j1*gdat.src.dim1]
            y3 = gdat.src.data[i1+1+(j1+1)*gdat.src.dim1]
            y4 = gdat.src.data[i1+(j1+1)*gdat.src.dim1]

            dsdx = ((1.0-u)*(y2-y1) + u*(y3-y4))/src_dx
            dsdy = ((1.0-t)*(y4-y1) + t*(y3-y2))/src_dy

            dd1[i1+j1*gdat.src.dim1] = dsdx
            dd2[i1+j1*gdat.src.dim1] = dsdy

            # Minus sign is needed!

            SO[(i+j*gdat.img.dim1),2*(i+j*gdat.img.dim1)]   = -dsdx
            SO[(i+j*gdat.img.dim1),2*(i+j*gdat.img.dim1)+1]= -dsdy

        else:

            if ((i1 == (gdat.src.dim1-1)) and (j1 in range(gdat.src.dim2-2)) and (t ==0.0)):

                # This is an array, not a grid, so indexing is different
```

```

                    y1 = gdat.src.data[i1+j1*gdat.src.dim1]
                    y4 = gdat.src.data[i1+(j1+1)*gdat.src.dim1]

                    dsdx = 0.0
                    dsdy = (y4-y1)/src_dy

                    dd1[i1+j1*gdat.src.dim1] = dsdx
                    dd2[i1+j1*gdat.src.dim1] = dsdy

                    # Minus sign is needed!

                    SO[(i+j*gdat.img.dim1),2*(i+j*gdat.img.dim1)] = -dsdx
                    SO[(i+j*gdat.img.dim1),2*(i+j*gdat.img.dim1)+1]= -dsdy

                if ((j1 == (gdat.src.dim2-1)) and (i1 < (gdat.src.dim1-1)) and (u == 0.0)):

                    # This is an array, not a grid, so indexing is different

                    y1 = gdat.src.data[i1+j1*gdat.src.dim1]
                    y2 = gdat.src.data[i1+1+j1*gdat.src.dim1]

                    dsdx = (y2-y1)/src_dx
                    dsdy = 0.0

                    dd1[i1+j1*gdat.src.dim1] = dsdx
                    dd2[i1+j1*gdat.src.dim1] = dsdy

                    # Minus sign is needed!

                    SO[(i+j*gdat.img.dim1),2*(i+j*gdat.img.dim1)] = -dsdx
                    SO[(i+j*gdat.img.dim1),2*(i+j*gdat.img.dim1)+1]= -dsdy


                if ((j1 == (gdat.src.dim2-1)) and (i1 == (gdat.src.dim1-1))\
                        and (u == 0.0) and (t == 0.0)):

                    SO[(i+j*gdat.img.dim1),2*(i+j*gdat.img.dim1)] = 0.0
                    SO[(i+j*gdat.img.dim1),2*(i+j*gdat.img.dim1)+1]= 0.0


    ########## Second determine the d-potential derivative matrix ##########

```

```
        DO = spmatrix.ll_mat(2*gdat.img.dim1*gdat.img.dim2, gdat.gpot.dim1*gdat.gpot.dim2)

        for i in range(1,gdat.img.dim1-1):
            for j in range(1,gdat.img.dim2-1):

                xx = i*img_dx + gdat.img.xmin
                yy = j*img_dy + gdat.img.ymin

                tmp = deflect_info_grid(gdat,xx,yy)

                val1=tmp[0]
                val2=tmp[1]
                val3=tmp[2]

                i1 = val3[0]
                j1 = val3[1]

                if (i1 in range(1,gdat.gpot.dim1-2) and
                    j1 in range(1,gdat.gpot.dim2-2)):

                    DO[2*(i+j*gdat.img.dim1),(i1-1+j1*gdat.gpot.dim1)]           = val1[0]
                    DO[2*(i+j*gdat.img.dim1),(i1-1+(j1+1)*gdat.gpot.dim1)]       = val1[1]
                    DO[2*(i+j*gdat.img.dim1),(i1+j1*gdat.gpot.dim1)]             = val1[2]
                    DO[2*(i+j*gdat.img.dim1),(i1+(j1+1)*gdat.gpot.dim1)]         = val1[3]
                    DO[2*(i+j*gdat.img.dim1),(i1+1+j1*gdat.gpot.dim1)]           = val1[4]
                    DO[2*(i+j*gdat.img.dim1),(i1+1+(j1+1)*gdat.gpot.dim1)]       = val1[5]
                    DO[2*(i+j*gdat.img.dim1),(i1+2+j1*gdat.gpot.dim1)]           = val1[6]
                    DO[2*(i+j*gdat.img.dim1),(i1+2+(j1+1)*gdat.gpot.dim1)]       = val1[7]

                    DO[2*(i+j*gdat.img.dim1)+1,(i1+(j1-1)*gdat.gpot.dim1)]       = val2[0]
                    DO[2*(i+j*gdat.img.dim1)+1,(i1+j1*gdat.gpot.dim1)]           = val2[1]
                    DO[2*(i+j*gdat.img.dim1)+1,(i1+(j1+1)*gdat.gpot.dim1)]       = val2[2]
                    DO[2*(i+j*gdat.img.dim1)+1,(i1+(j1+2)*gdat.gpot.dim1)]       = val2[3]
                    DO[2*(i+j*gdat.img.dim1)+1,(i1+1+(j1-1)*gdat.gpot.dim1)]     = val2[4]
                    DO[2*(i+j*gdat.img.dim1)+1,(i1+1+j1*gdat.gpot.dim1)]         = val2[5]
                    DO[2*(i+j*gdat.img.dim1)+1,(i1+1+(j1+1)*gdat.gpot.dim1)]     = val2[6]
                    DO[2*(i+j*gdat.img.dim1)+1,(i1+1+(j1+2)*gdat.gpot.dim1)]     = val2[7]
```

```
    ########## Third matrix-multiply the two matrices ##########

    CO       = spmatrix.ll_mat(gdat.img.dim1*gdat.img.dim2, gdat.
        src.dim1*gdat.src.dim2 + gdat.gpot.dim1*gdat.gpot.dim2)
    CO       = spmatrix.matrixmultiply(SO,DO)

    ########## Now enter CO in to SPO ##########

    kl=gdat.src.dim1*gdat.src.dim2
    mn=gdat.gpot.dim1*gdat.gpot.dim2
    pq=gdat.img.dim1*gdat.img.dim2

    SPO[0:pq,kl:kl+mn] = CO[0:pq,0:mn]

##########################################
# Determine pmask
##########################################

for i in range(gdat.gpot.dim1):
    for j in range(gdat.gpot.dim2):

        xx = i*gpot_dx + gdat.gpot.xmin
        yy = j*gpot_dy + gdat.gpot.ymin

        i_d = int(floor((xx-gdat.img.xmin)/img_dx))
        j_d = int(floor((yy-gdat.img.ymin)/img_dy))

        sv = source_pos(ldat1,ldat2,gdat,xx,yy,fl_dpot)

        i1 = int(floor((sv[0]-gdat.src.xmin)/src_dx))
        j1 = int(floor((sv[1]-gdat.src.ymin)/src_dy))

        t = (sv[0] - (i1*gpot_dx + gdat.gpot.xmin))/src_dx
        u = (sv[1] - (j1*gpot_dy + gdat.gpot.ymin))/src_dy

        cxi = corr_x_i(t,i1)
        t   = cxi[0]
        i1 = cxi[1]

        cxi = corr_x_i(u,j1)
        u   = cxi[0]
        j1 = cxi[1]

        # if pixel is inside source-plane grid, then continue

        # there should also be data!

        if (i1 in range(gdat.src.dim1-1) and \
            j1 in range(gdat.src.dim2-1) and \
            data_mask[i_d+j_d*gdat.img.dim1] == 1.0):

            pmask[i+j*gdat.gpot.dim1]          = 1.0

```

```
    # ##############################################################################
    # The SPO operator has been determine and can be returned
    # ##############################################################################

    SPO_conv = spmatrix.ll_mat(gdat.img.dim1*gdat.img.dim2, gdat.src.dim1*gdat.src.dim2 \
                               + gdat.gpot.dim1*gdat.gpot.dim2)
    SPO_conv = spmatrix.matrixmultiply(BO,SPO)

    return SPO, lmask, smask, pmask


# ##############################################################################
# Read psf fits files
# ##############################################################################

def read_psf():

    # open psf fits-file

    hdulist = pyfits.open("psf.fits")
    psf_tmp = hdulist[0].data

    dim1    = psf_tmp.shape[0]

    dim2    = psf_tmp.shape[1]

    tmpdat = psf_tmp

    # read data and array scale

    data_psf = zeros(dim1*dim2,'d')

    for i in range(dim1):
        for j in range(dim2):
            data_psf[i+j*dim1]=tmpdat[j,i]

    return data_psf, dim1, dim2


# ##############################################################################
# Convolution Operator
```

```
# ######################################################################################

def convop(gdat):

    #print 'Determining Convolution Operator'

    BO = spmatrix.ll_mat(gdat.img.dim1*gdat.img.dim2, gdat.img.dim1*gdat.img.dim2)

    sum=0.0

    for ii in range(-(gdat.psf.size-1)/2,1+(gdat.psf.size-1)/2):
        for jj in range(-(gdat.psf.size-1)/2,1+(gdat.psf.size-1)/2):

            sum += gdat.psf.data[ii+gdat.psf.cx+(jj+gdat.psf.cy)*gdat.psf.dim1]


    for i in range(gdat.img.dim1):

        for j in range(gdat.img.dim2):

            for ii in range(-(gdat.psf.size-1)/2,1+(gdat.psf.size-1)/2):
                for jj in range(-(gdat.psf.size-1)/2,1+(gdat.psf.size-1)/2):

                    i1 = ii+i
                    j1 = jj+j

                    if (i1 in range(gdat.img.dim1) and
                        j1 in range(gdat.img.dim2)):

                        # This is an array, not a grid, so indexing is different

                        BO[i+j*gdat.img.dim1, i1+j1*gdat.img.dim1] = \
                                                 gdat.psf.data[ii+gdat.psf.cx+\
                                                          (jj+gdat.psf.cy)*gdat.psf.dim1]/sum


    return BO

# ######################################################################################
```

```
# Create source
# ####################################################################################

def src_img_create(ldat1,ldat2,gdat,fl_dpot,sig,q,pa,sx0,sy0,sig2,q2,
    pa2,sx02,sy02,ratio,BO,flag):

    img_dx = (gdat.img.xmax-gdat.img.xmin)/(gdat.img.dim1-1.0)
    img_dy = (gdat.img.ymax-gdat.img.ymin)/(gdat.img.dim2-1.0)

    # Create source grid

    src_dx = (gdat.src.xmax-gdat.src.xmin)/(gdat.src.dim1-1.0)
    src_dy = (gdat.src.ymax-gdat.src.ymin)/(gdat.src.dim2-1.0)

    data  = zeros(gdat.src.dim1*gdat.src.dim2,'d')
    data2 = zeros(gdat.img.dim1*gdat.img.dim2,'d')

    lmask_orig = ones(gdat.img.dim1*gdat.img.dim2,'d')

    for i in range(gdat.src.dim1):
        for j in range(gdat.src.dim2):

            # (i,j) corresponds to physical scale (xx,yy)
            #  i -> row
            #  j -> col

            # SRC1

            xx = i*src_dx + gdat.src.xmin
            yy = j*src_dy + gdat.src.ymin

            xx=xx-sx0
            yy=yy-sy0

            tr = pi*(pa/180.0)+pi/2.0

            cs=cos(tr)
            sn=sin(tr)

            sx_r =  xx*cs+yy*sn
            sy_r = -xx*sn+yy*cs

            if flag == 0: #standard peak brightness of source
              data[i+j*gdat.src.dim1] = 100.0*exp(-((((sx_r)**2.0+((
                  sy_r)/q)**2.0))/(sig**2.0))**0.5)
            elif flag == 1: #flux normalized source
              data[i+j*gdat.src.dim1] = (100.0/(2.*m.pi*(sig**2.)))*
                  exp(-((((sx_r)**2.0+((sy_r)/q)**2.0))/(sig**2.0))
                  **0.5)

            # SRC2

```

```
            xx2 = i*src_dx + gdat.src.xmin
            yy2 = j*src_dy + gdat.src.ymin

            xx2=xx2-sx02
            yy2=yy2-sy02

            tr2 = pi*(pa2/180.0)+pi/2.0

            cs2=cos(tr2)
            sn2=sin(tr2)

            sx_r2 =  xx2*cs2+yy2*sn2
            sy_r2 = -xx2*sn2+yy2*cs2

            data[i+j*gdat.src.dim1] = data[i+j*gdat.src.dim1] #+ \
                                      #100.0*ratio*exp(-((((sx_r2)
                                          **2.0+(sy_r2/q2)**2.0))/(sig2
                                          **2.0))**0.5)



    # create lensed source grid

    LO = source_psi_op(ldat1,ldat2,gdat,lmask_orig,BO,fl_dpot,0)

    vec_tmp = zeros(gdat.src.dim1*gdat.src.dim2+gdat.gpot.dim1*gdat.
        gpot.dim2,'d')
    vec_tmp[0:gdat.src.dim1*gdat.src.dim2] = data[0:gdat.src.dim1*gdat
        .src.dim2]

    LO[0].matvec(vec_tmp,data2)

    # return source and convolved image and image mask

    return data,data2,LO[1],LO[2]

#
    ################################################################################
# Add two ll_mat matrices
#
    ################################################################################

def add_ll_mat(A,B):

    assert A.shape == B.shape

    C = A.copy()
    C.shift(1.0,B)

    return C

```

```
# ################################################################################
# Identity matrix
# ################################################################################

def regul_ll_mat1(d1,d2,lamb):

    I = spmatrix.ll_mat(d1*d2,d1*d2)

    for i in range(d1*d2):
        I[i,i]=lamb

    return I

def regul_ll_mat2(d1,d2,lamb):

    val = [-1.0, 3.0, -3.0, 1.0]

    T = spmatrix.ll_mat(d1*d2,d1*d2)
    I = spmatrix.ll_mat(d1*d2,d1*d2)

    for i in range(d1-len(val)):
        for j in range(d2):

            for l in range(len(val)):
                n1=i+j*d1
                n2=(i+l)+j*d1
                T[n1,n2]=sqrt(lamb)*val[l]

    for i in range(d1-len(val),d1):
        for j in range(d2):

            for l in range(len(val)):
                n1=i+j*d1
                n2=(i-l)+j*d1
                T[n1,n2]=sqrt(lamb)*val[l]

    I = spmatrix.dot(T,T)

    return I

def regul_ll_mat3(d1,d2,lamb):

    val = [-1.0, 3.0, -3.0, 1.0]

    T = spmatrix.ll_mat(d1*d2,d1*d2)
    I = spmatrix.ll_mat(d1*d2,d1*d2)

    for i in range(d1):
        for j in range(d2-len(val)):

```

```
            for l in range(len(val)):
                n1=i+j*d1
                n2=i+(j+l)*d1
                T[n1,n2]=sqrt(lamb)*val[l]

    for i in range(d1):
        for j in range(d2-len(val),d2):

            for l in range(len(val)):
                n1=i+j*d1
                n2=i+(j-l)*d1
                T[n1,n2]=sqrt(lamb)*val[l]

    I = spmatrix.dot(T,T)

    return I

# ################################################################################
# Set up linear system with regularisation -> large sparse matrix SPO^T.SPO + R
# ################################################################################

def sol_matrix(SPO,gdat,lamb1_rms,lamb1_drv,lamb2_rms,lamb2_drv):

    # chi^2 matrix -> (mn+kl)x(mn+kl)

    M1 = spmatrix.dot(SPO,SPO)   # =>  SPO^T.SPO

    kl=gdat.src.dim1*gdat.src.dim2
    mn=gdat.gpot.dim1*gdat.gpot.dim2

    # regularisation matrices for source

    rm1 = regul_ll_mat1(gdat.src.dim1,gdat.src.dim2,lamb1_rms)
    rm2 = regul_ll_mat2(gdat.src.dim1,gdat.src.dim2,lamb1_drv)
    rm3 = regul_ll_mat3(gdat.src.dim1,gdat.src.dim2,lamb1_drv)

    # -> klxkl

    R1 = add_ll_mat(add_ll_mat(rm1,rm2),rm3)

    # regularisation matrices for dpot

    rm4 = regul_ll_mat1(gdat.gpot.dim1,gdat.gpot.dim2,lamb2_rms)
    rm5 = regul_ll_mat2(gdat.gpot.dim1,gdat.gpot.dim2,lamb2_drv)
    rm6 = regul_ll_mat3(gdat.gpot.dim1,gdat.gpot.dim2,lamb2_drv)

    # -> mnxmn

    R2 = add_ll_mat(add_ll_mat(rm4,rm5),rm6)
```

```
    # Block matrix for regularisation

    M2 = spmatrix.ll_mat(mn+kl,mn+kl)

    # Now substitute R1 and R2 as block-diagonal matrixes in to M2

    M2[0:kl,0:kl]           = R1[0:kl,0:kl]
    M2[kl:kl+mn,kl:kl+mn]   = R2[0:mn,0:mn]

    M = add_ll_mat(M1,M2)

    return M

#
    ################################################################################
# Set up linear system, vector SPO^T.vec(d)
#
    ################################################################################

def sol_vector(SPO,data):

    temp = zeros(SPO.shape[1],'d')
    SPO.matvec_transp(data,temp)
    return temp

#
    ################################################################################
# Fit a plane to three corner points and subtract from grid
#
    ################################################################################

def fplane(gdat,SS2):

    SOL = zeros(gdat.gpot.dim1*gdat.gpot.dim2,'d')

    # three corners that should be zero

    psi1 = SS2[0]
    psi2 = SS2[(gdat.gpot.dim1-1)]
    psi3 = SS2[(gdat.gpot.dim2-1)*gdat.gpot.dim1]

    p0 = array([0.0,0.0,psi1])
    v1 = array([1.0,0.0,psi2-psi1])
    v2 = array([0.0,1.0,psi3-psi1])

    for i in range(gdat.gpot.dim1):
        for j in range(gdat.gpot.dim2):

            s = 1.0*i/(gdat.gpot.dim1-1.0)
```

```
            t = 1.0*j/(gdat.gpot.dim2-1.0)

            psi_est = psi1 + s*(psi2-psi1) + t*(psi3-psi1)
            SOL[i+j*gdat.gpot.dim1] = SS2[i+j*gdat.gpot.dim1] - psi_est

    return SOL


def fplane_2(gdat,pmask,SS2):

    gpot_dx = (gdat.gpot.xmax - gdat.gpot.xmin)/(gdat.gpot.dim1-1)
    gpot_dy = (gdat.gpot.ymax - gdat.gpot.ymin)/(gdat.gpot.dim2-1)

    SOL = zeros(gdat.gpot.dim1*gdat.gpot.dim2,'d')

    # determine average x,y gradients

    gx = 0.0
    gy = 0.0
    pt = 0.0

    nuls = 0

    for i in range(gdat.gpot.dim1):
        for j in range(gdat.gpot.dim2):

            if (pmask[i+j*gdat.gpot.dim1] > 0.5):

                xx = i*gpot_dx+gdat.gpot.xmin
                yy = j*gpot_dy+gdat.gpot.ymin
                da = deflect_dpot(gdat, SS2, xx, yy)

                if (da[0]!=0.0 and da[1]!=0.0):
                    gx = gx + da[0]
                    gy = gy + da[1]
                else:
                    nuls = nuls + 1

    if (nuls != sum(pmask)):
        gx = gx/(sum(pmask)-nuls)
        gy = gy/(sum(pmask)-nuls)

    # three corners that should be zero -- psi1 is on (0,0) as reference corner
    # although any point could have been chosen

    psi2 = gx*(gdat.gpot.xmax-gdat.gpot.xmin)
    psi3 = gy*(gdat.gpot.ymax-gdat.gpot.ymin)

    for i in range(gdat.gpot.dim1):
        for j in range(gdat.gpot.dim2):

            s = (1.0*i)/(gdat.gpot.dim1-1.0)
```

```python
            t = (1.0*j)/(gdat.gpot.dim2-1.0)
            psi_est = s*psi2 + t*psi3
            SOL[i+j*gdat.gpot.dim1] = SS2[i+j*gdat.gpot.dim1] - psi_est

    pt  = sum(SOL*pmask)/sum(pmask)
    SOL = SOL - pt

    return SOL


# ###############################################################################
# Main body
# ###############################################################################

def main(noi,sigpow,argv=sys.argv):

    class lensdata:  # all lens data
        pass

    class griddata:  # all grid(s) data
        class psf:
            pass
        class src:
            pass
        class img:
            pass
        class gpot:
            pass

    ##########################################

    ldat1 = lensdata()
    ldat2 = lensdata()
    gdat  = griddata()

    ##########################################

    gdat.src.dim1 =   80
    gdat.src.dim2 =   80
    gdat.src.xmin = -1.0
    gdat.src.xmax =  1.0
    gdat.src.ymin = -1.0
    gdat.src.ymax =  1.0


    ##########################################

    # image grid

```

```
    gdat.img.dim1 =  80
    gdat.img.dim2 =  80
    gdat.img.xmin = -2.00
    gdat.img.xmax =  2.00
    gdat.img.ymin = -2.00
    gdat.img.ymax =  2.00

    # potential grid

    gdat.gpot.dim1 =  80
    gdat.gpot.dim2 =  80
    gdat.gpot.xmin = -2.00
    gdat.gpot.xmax =  2.00
    gdat.gpot.ymin = -2.00
    gdat.gpot.ymax =  2.00

    ##########################################

    # lens 1 data
    ldat1.bl = 0.5
    ldat1.th =  0.00
    ldat1.fl =  0.85
    ldat1.x0 =  0.00
    ldat1.y0 =  0.00
    ldat1.rc =  1.0e-4
    ldat1.ss =  0.000
    ldat1.sa =  0.000


    # lens 2 data
    ldat2.bl =  1.0e-10
    ldat2.th = 0.00
    ldat2.fl =  0.999
    ldat2.x0 = -0.9
    ldat2.y0 = -0.4
    ldat2.rc =  1.0e-4
    ldat2.ss =  0.0
    ldat2.sa =  0.0

    ##########################################

    # get ACS PSF

    tmp = read_psf()
    gdat.psf.data=tmp[0]
    gdat.psf.dim1=tmp[1]
    gdat.psf.dim2=tmp[2]
    gdat.psf.cx = 37
    gdat.psf.cy = 37

    gdat.psf.size = 7 #15

    # determine convolution operator

```

```
    BO = spmatrix.ll_mat(gdat.img.dim1*gdat.img.dim2, gdat.img.dim1*gdat.img.dim2)
    BO = convop(gdat)


    ###########################################
    #Varying parameters
    ###########################################

    fields = 100 #total number of simulations
    gdat.gpot.fluctsig = 10.**(float(sigpow)) #variance of the random field fluctuations
    noise = noi #setting the noise level
    gdat.src.sig = 0.25 #source rms
    flag = 0 #flag = 0 for same peak brightness of source, flag = 1 for flux normalized source
    pspec = -4. #potential fluctuations power spectrum slope

    ###########################################

    for gen in range(fields):
        #create empty potential data vectors
        gdat.gpot.data   = pot_grid(gdat,ldat1,ldat2,gen,gdat.src.sig,noise,sigpow,pspec)

        #create source and lensed image, parameters:
        data_all = src_img_create(ldat1,ldat2,gdat,1,\
                                  gdat.src.sig,1.0,0.0,0.00,0.20,\
                                  0.0,0.999,0.0,-0.40,0.25,0.5,BO,flag)

        # add noise to the lensed image
        gdat.img.data = data_all[1] +\
                        array(ranlib.normal(0.0,noise,[gdat.img.dim2*gdat.img.dim1]))-min(data_all[0])

        # keep original source and lensed image

        imag_orig  = gdat.img.data
        lmask_orig = data_all[2]

        # create empty source data vectors

        gdat.src.data    = data_all[0]

        kappa            = convergence(gdat,ldat1,ldat2)

        ###########################################

        dmask = zeros(gdat.img.dim1*gdat.img.dim2,'d')

        for i in range(gdat.img.dim1):
            for j in range(gdat.img.dim2):
```

```
                    if (gdat.img.data[i+j*gdat.img.dim2]>=-100000.0): # 3.0*noise):
                        dmask[i+j*gdat.img.dim2] = 1.0

            pyfits.writeto('./generations/src_'+str(gdat.src.sig)+'/10'+str(sigpow)+'nois'+str(noise)+'/4sim_lns'+str(gen+1)+'.fits',reshape(gdat.img.data,[gdat.img.dim1,gdat.img.dim2]))
            pyfits.writeto('./generations/src_'+str(gdat.src.sig)+'/10'+str(sigpow)+'nois'+str(noise)+'/4sim_pot'+str(gen+1)+'.fits',reshape(gdat.gpot.data,[gdat.gpot.dim1,gdat.gpot.dim2]))

    histo(fields,gdat.gpot.fluctsig,gdat.src.sig,noise,sigpow) # creating distribution of all random fields
    residualscreate(fields,gdat.src.sig,noise,sigpow) #creating residuals
    pspecall(fields,gdat.gpot.fluctsig,noise,gdat.src.sig,sigpow) # generating power spectra from the residuals

# run this code

noises = [5.0,12.0,100.0]
sigpow = [-3,-4,-5]

for si in sigpow:
  for no in noises:
    print 'using noise = ',no,', sigma^2 = 10^',si,', source size = 0.8'
    main(no,si)


print 'finished!'
```

Appendix C

Gaussian Random Field Code

```
#! /usr/bin/env python

import numpy as n
import math as m
import random as ran
import pyfits
from matplotlib import pyplot as plt

#####################################################################
#Power spectrum
#####################################################################
def powspec(L,variance,Npix,Psum,power):
  if L == 0.0:
    P = 0.0
  else:
    A = variance*(Npix**2.)/(2.*Psum)
    P = A*L**(power)
  return P

#####################################################################
#Sum over the power spectrum
#####################################################################
def Psum_calculator(dimlx,dimly,Lx,Ly,power):
  lxaxis = n.append(n.arange(0.,(dimlx/2.)/Lx,1./Lx),n.arange((-dimlx
      /2.)/Lx,0.,1./Lx))
  lyaxis = n.append(n.arange(0.,(dimly/2.)/Ly,1./Ly),n.arange((-dimly
      /2.)/Ly,0.,1./Ly))

  lx = list(n.zeros([dimlx,1]))
  ly = list(n.zeros([dimly,1]))

  for x in range(len(lx)):
    lx[x] = lxaxis

  for y in range(len(ly)):
    ly[y] = lyaxis

  lx = n.array(lx)
```

```
  ly = n.transpose(n.array(ly))
  l = n.sqrt(lx**2. + ly**2.)


  summ = 0.
  for y in range(n.shape(l)[0]):
    for x in range(n.shape(l)[1]):
      if l[y][x] == 0.:
        summ += 0.
      else:
        summ += l[y][x]**(power)
  return summ

####################################################################
#Creating a Fourier grid
####################################################################
def fourierplane(a,power):

  j = 0 + 1j #redefining complex number 1j for use later on

  plane = n.zeros([a.dimlx,a.dimly],dtype='cfloat') #Empty matrix to
      be filled in for the Fourier Plane

  lxaxis = n.append(n.arange(0.,(a.dimlx/2.)/a.Lx,1./a.Lx),n.arange((-
      a.dimlx/2.)/a.Lx,0.,1./a.Lx))
  lyaxis = n.append(n.arange(0.,(a.dimly/2.)/a.Ly,1./a.Ly),n.arange((-
      a.dimly/2.)/a.Ly,0.,1./a.Ly))

  Psum = Psum_calculator(a.dimlx,a.dimly,a.Lx,a.Ly,power)

  for y in range(n.shape(plane)[0]):
    for x in range(n.shape(plane)[1]):
      #Defining coordinates centred at x = N/2, y = N/2
      i1 = x - a.dimlx/2
      j1 = y - a.dimly/2

      #Determining coordinates in Fourier-space on the grid
      lx = lxaxis[x]
      ly = lyaxis[y]
      l = m.sqrt(lx**2. + ly**2.) #Magnitude of l-vector

      #Box-Muller transform, polar form:
      sigma = m.sqrt(powspec(l,a.varia,a.dimlx*a.dimly,Psum,power)) #
          Width of the Gaussian distribution
      s = 1.1
      while s > 1.:
        u = ran.uniform(-1.,1.)
        v = ran.uniform(-1.,1.)
        s = u**2. + v**2.
      fac = m.sqrt(-2.*m.log(s)/s)
      z1 = u*fac*sigma
      z2 = v*fac*sigma

      #Normal Box-Muller transform
```

```
      #u = ran.uniform(0,1)
      #v = ran.uniform(0,1)
      #fac = m.sqrt(-2.*m.log(u))
      #z1 = fac*m.cos(2.*m.pi*v)*sigma
      #z2 = fac*m.sin(2.*m.pi*v)*sigma

      ######################################################################
      #Filling in the grid
      ######################################################################
      if x == 0 and y == 0: #Gives the average of the field
        plane[y][x] = 0.0

      #Three points that need to be real-valued to get a real image after FFT:
      elif x == 0 and y == a.dimly/2:
        plane[y][x] = z1
      elif x == a.dimlx/2 and y == 0:
        plane[y][x] = z1
      elif x == a.dimlx/2 and y == a.dimly/2:
        plane[y][x] = z1

      else:
        plane[y][x] = z1 + j*z2

      ######################################################################
      #Creating symmetry f(k) = f*(-k)
      ######################################################################
      y2 = -(j1 + a.dimly/2)
      x2 = -(i1 + a.dimlx/2)
      plane[y2][x2] = plane[y][x].conjugate()

    if y > n.shape(plane)[0]/2.:
      break #Due to symmetry in grid, only the top half has to be filled in for completing the full grid

  return plane

######################################################################
#Setting up all the necessary parameters and running the code
######################################################################

def substruct(gpot,gen,sig,src_sig,noise,sigpow,power):

  class fougrid:
    pass

  grid = fougrid()

  grid.varia = sig #Variance (sigma^2) of the fluctuations

  #Defining parameters for the grid in Fourier-space
  grid.dimlx = gpot.dim1 #Dimensions in x-direction, same as the original lensed image
```

```
    grid.dimly = gpot.dim2 #Dimension in y-direction, " " " " " " " " " "

    grid.deltax = (gpot.xmax-gpot.xmin)/gpot.dim1 #Pixel size in x-direction in real space
    grid.deltay = (gpot.ymax-gpot.ymin)/gpot.dim2 #Pixel size in y-direction in real space

    grid.Lx = gpot.xmax-gpot.xmin #Size of the image in real space in x-direction
    grid.Ly = gpot.ymax-gpot.ymin #Size of the image in real space in y-direction

    fplane = fourierplane(grid,power) #Creating the Fourier plane

    implane = n.fft.ifftshift(n.fft.ifft2(fplane)) #Inverse Fourier transform of the Fourier plane to get the final image

    realimplane = implane.real #The final image, still some very small residuals in the imaginary part after the Fourier transform

    pyfits.writeto('./generations/src_'+str(src_sig)+'/10'+str(sigpow)+'nois'+str(noise)+'/randomfield'+str(gen+1)+'.fits',realimplane) #Saving the image to file

    return realimplane
```

Appendix D

Residuals and Power Spectrum Code

```python
#! /usr/bin/env python

import pyfits as pf
import numpy as n
import math as m
from matplotlib import pyplot as plt
from matplotlib import rc
rc('text', usetex=True)

######################################################################
#Subtracting two images from eachother
######################################################################
def subtr_image(im1,im2):

  hdu1 = pf.open(im1)
  hdu2 = pf.open(im2)

  data1 = hdu1[0].data
  data2 = hdu2[0].data

  subtr = data2 - data1
  return subtr

######################################################################
#Create residuals for all simulations
######################################################################
def residualscreate(fields,src_sig,noise,sigpow):

  image1 = str('./nofluct/4sim_lns0src_'+str(src_sig)+'.fits') # Simulation without potential fluctuations

  for x in range(fields):
    image2 = str('./generations/src_'+str(src_sig)+'/10'+str(sigpow)+'nois'+str(noise)+'/4sim_lns'+str(x+1)+'.fits')
    new = subtr_image(image1,image2)
```

```python
    pf.writeto('./generations/src_'+str(src_sig)+'/10'+str(sigpow)+'
        nois'+str(noise)+'/res_'+str(x+1)+'.fits',new)

####################################################################
#Determining the power spectra
####################################################################
def pspecall(fields,noise,src_sig,sigpow):#determine the desir

  pspecs = [] #List to be filled with the powerspectra

  for b in range(fields):
    hdu = pf.open('./generations/src_'+str(src_sig)+'/10'+str(sigpow)+
        'nois'+str(noise)+'/res_'+str(b+1)+'.fits') #Open all residuals
         files
    data = hdu[0].data

    fouriertf = n.fft.fftshift(n.fft.fft2(n.fft.ifftshift(data))) #
        Fast Fourier transforming the residuals
    absval2 = fouriertf.real**2. + fouriertf.imag**2 #Take the
        absolute value squared

    steps = 10 #number of bins in which to determine the powerspectrum

    #Dimensions of the Fourier transformed image
    diml_x = n.shape(absval2)[1]
    diml_y = n.shape(absval2)[0]

    #Physical lengths in real space of the image (need to be same as
        in lensing code)
    L_x = 4.0
    L_y = 4.0

    steplist = range(steps+1)

    pspeclist = n.array([]) #List that will be filled with the
        powerspectrum value at every l in the grid

    #Axes in Fourier space
    l_xlist = n.arange((-diml_x/2.)/L_x,(diml_x/2)/L_x,1./L_x)
    l_ylist = n.arange((-diml_y/2.)/L_y,(diml_y/2)/L_y,1./L_y)

    lmax = m.sqrt(n.max(n.abs(l_xlist))**2. +n.max(n.abs(l_ylist))
        **2.)

    #Generating the powerspectra
    for step in range(steps):
      bin = n.array([])

      for x in range(diml_x):
        for y in range(diml_y):
          lx = l_xlist[x]
          ly = l_ylist[y]
          l = m.sqrt(lx**2. + ly**2.)

```

```
                if steplist[step]*lmax/steps < l <= steplist[step+1]*lmax/steps:
                    bin = n.append(bin,absval2[y][x])
            pspeclist = n.append(pspeclist,n.mean(bin)) #Adding each bin value to pspeclist
        pspecs.append(pspeclist) #Adding the entire powerspectrum to the list

    l_list = n.linspace(lmax/(2.*steps),lmax-lmax/(2.*steps),steps) #List for the x-axis of the plots, values are set halfway each bin
    n.save('l_list',l_list) #Saving the x-axis to a file
    n.save('powerspectrasrc_'+str(src_sig)+'10-'+str(sigpow)+'nois'+str(noise),pspecs) #Saving the powerspectra to a file
```

Appendix E

Statistics and Plotting Code

```
import pyfits as pf
import numpy as n
import math as m
from matplotlib import pyplot as plt
from matplotlib import rc
rc('text', usetex=True)

######################################################################
#Some general plotting parameters
######################################################################
def plotstuff(flagx,flagy):
  plt.xscale('log')
  plt.yscale('log')

  plt.xticks(fontsize=20)
  plt.yticks(fontsize=20)

  if flagx == 1:
    plt.xlabel(r'$\frac{k}{2\pi}$ in $arcsec^{-1}$')
  if flagy == 1:
    plt.ylabel(r'P($\frac{k}{2\pi}$)')

######################################################################
#Calculation of errors and mean spectrum
######################################################################
def datacal(sigpow,noise,src_sig):

  pspecs = n.load('powerspectrasrc_'+str(src_sig)+'10-'+str(sigpow)+'nois'+str(noise)+'.npy') #Load power spectra from file

  means = n.array([]) #Array that will be filled with the mean power spectrum

  #Array that will be filled with errors
  err1 = n.array([])

  for lbin in range(len(pspecs[0])):
```

```
        mean = n.mean(pspecs[[slice(None),]+[lbin]]) #Calculate the mean value of every bin
        means = n.append(means, mean)

        rms2bin = n.sum((pspecs[[slice(None),]+[lbin]]-mean)**2.)/len(pspecs[0]) #Calculating the rms^2 in a single bin
        err1 = n.append(err1,m.sqrt(rms2bin))

    err100 = err1/m.sqrt(100.) #Errors after 100 measurements

    return means, err1, err100

########################################################################
#Calculate chi^2 values
########################################################################
def chi2(noise,meanspec,error1):
    noisespec = n.ones(n.shape(meanspec))*80*80*(noise**2.)
    chi2 = n.sum((((meanspec-noisespec)/error1)**2.))

    return chi2


########################################################################
#Plotting in subfigures
########################################################################
llist = n.load('l_list.npy') #Loading x-axis file for all plots

    f_handle = file('chisquared'+str(src_sig)+'.txt','a') #open a chisquared text file in append mode

    plt.figure()

    plt.subplot(3,3,1)
    noise = 5.0 #noise level
    sigpow = -3 #fluctuation scale: sigma^2 = 10^sigpow

    data = datacal(sigpow,noise,src_sig) #get mean power spectrum and errors

    plt.errorbar(llist,data[0],yerr=data[1],fmt='.', ecolor='b',label = 'Error for N=1') #plot with error for single measurement
    plt.errorbar(llist,data[0],yerr=data[2],fmt='.', ecolor='r',label = 'Error for N=100') #plot with error for 100 measurements

    plt.plot(llist,n.ones(n.shape(llist))*80*80*(noise**2.),'g-',label='Noise level power spectrum') #plot noise power spectrum

    plt.figtext(0.25,0.8,'Noise level: 5.0')
    plotstuff(0,1)
    plt.ylim(4.*(10.**3.),3.*(10.**7.))
    plt.title(r'$\sigma_{fluct}^2 = 10^{-3}$',fontsize=24)

    n.savetxt(f_handle,chi2(noise,data[0],data[1])) #add chi squared value to file
```

```
plt.subplot(3,3,2)
noise = 5.0
sigpow = -4
data = datacal(sigpow,noise,src_sig)
plt.errorbar(llist,data[0],yerr=data[1],fmt='.', ecolor='b')
plt.errorbar(llist,data[0],yerr=data[2],fmt='.', ecolor='r')
plt.plot(llist,n.ones(n.shape(llist))*80*80*(noise**2.),'g',label='Noise level power spectrum')
#plt.figtext(0.25,0.5,'Noise level: '+str(noise))
plotstuff(0,0)
plt.ylim(4.*(10.**3.),3.*(10.**7.))
plt.title(r'$\sigma_{fluct}^2 = 10^{-4}$',fontsize=24)
n.savetxt(f_handle,chi2(noise,data[0],data[1]))

plt.subplot(3,3,3)
noise = 5.0
sigpow = -5
data = datacal(sigpow,noise,src_sig)
plt.errorbar(llist,data[0],yerr=data[1],fmt='.', ecolor='b')
plt.errorbar(llist,data[0],yerr=data[2],fmt='.', ecolor='r')
plt.plot(llist,n.ones(n.shape(llist))*80*80*(noise**2.),'g',label='Noise level power spectrum')
#plt.figtext(0.25,0.5,'Noise level: '+str(noise))
plotstuff(0,0)
plt.ylim(4.*(10.**3.),3.*(10.**7.))
plt.title(r'$\sigma_{fluct}^2 = 10^{-5}$',fontsize=24)
n.savetxt(f_handle,chi2(noise,data[0],data[1]))

plt.subplot(3,3,4)
noise = 12.0
sigpow = -3
data = datacal(sigpow,noise,src_sig)
plt.errorbar(llist,data[0],yerr=data[1],fmt='.', ecolor='b')
plt.errorbar(llist,data[0],yerr=data[2],fmt='.', ecolor='r')
plt.plot(llist,n.ones(n.shape(llist))*80*80*(noise**2.),'g',label='Noise level power spectrum')
plt.figtext(0.25,0.55,'Noise level: 12.0')
plotstuff(0,1)
plt.ylim(9.*(10.**4.),2.*(10.**7.))
n.savetxt(f_handle,chi2(noise,data[0],data[1]))

plt.subplot(3,3,5)
noise = 12.0
sigpow = -4
data = datacal(sigpow,noise,src_sig)
plt.errorbar(llist,data[0],yerr=data[1],fmt='.', ecolor='b',label = 'Error for N=1')
plt.errorbar(llist,data[0],yerr=data[2],fmt='.', ecolor='r',label = 'Error for N=100')
plt.plot(llist,n.ones(n.shape(llist))*80*80*(noise**2.),'g',label='Noise level power spectrum')
#plt.figtext(0.25,0.5,'Noise level: '+str(noise))
plotstuff(0,0)
```

```
plt.ylim(9.*(10.**4.),2.*(10.**7.))
plt.legend()
n.savetxt(f_handle,chi2(noise,data[0],data[1]))

plt.subplot(3,3,6)
noise = 12.0
sigpow = -5
data = datacal(sigpow,noise,src_sig)
plt.errorbar(llist,data[0],yerr=data[1],fmt='.', ecolor='b')
plt.errorbar(llist,data[0],yerr=data[2],fmt='.', ecolor='r')
plt.plot(llist,n.ones(n.shape(llist))*80*80*(noise**2.),'g',label='
    Noise level power spectrum')
#plt.figtext(0.25,0.5,'Noise level: '+str(noise))
plotstuff(0,0)
plt.ylim(9.*(10.**4.),2.*(10.**7.))
n.savetxt(f_handle,chi2(noise,data[0],data[1]))

plt.subplot(3,3,7)
noise = 100.0
sigpow = -3
data = datacal(sigpow,noise,src_sig)
plt.errorbar(llist,data[0],yerr=data[1],fmt='.', ecolor='b')
plt.errorbar(llist,data[0],yerr=data[2],fmt='.', ecolor='r')
plt.plot(llist,n.ones(n.shape(llist))*80*80*(noise**2.),'g',label='
    Noise level power spectrum')
plt.figtext(0.25,0.15,'Noise level: 100.0')
plotstuff(1,1)
plt.ylim(3.*(10.**7.),1.2*(10.**8.))
n.savetxt(f_handle,chi2(noise,data[0],data[1]))

plt.subplot(3,3,8)
noise = 100.0
sigpow = -4
data = datacal(sigpow,noise,src_sig)
plt.errorbar(llist,data[0],yerr=data[1],fmt='.', ecolor='b')
plt.errorbar(llist,data[0],yerr=data[2],fmt='.', ecolor='r')
plt.plot(llist,n.ones(n.shape(llist))*80*80*(noise**2.),'g',label='
    Noise level power spectrum')
#plt.figtext(0.25,0.5,'Noise level: '+str(noise))
plotstuff(1,0)
plt.ylim(3.*(10.**7.),1.2*(10.**8.))
n.savetxt(f_handle,chi2(noise,data[0],data[1]))

plt.subplot(3,3,9)
noise = 100.0
sigpow = -5
data = datacal(sigpow,noise,src_sig)
plt.errorbar(llist,data[0],yerr=data[1],fmt='.', ecolor='b')
plt.errorbar(llist,data[0],yerr=data[2],fmt='.', ecolor='r')
plt.plot(llist,n.ones(n.shape(llist))*80*80*(noise**2.),'g',label='
    Noise level power spectrum')
#plt.figtext(0.25,0.5,'Noise level: '+str(noise))
plotstuff(1,0)
plt.ylim(3.*(10.**7.),1.2*(10.**8.))
```

```
    n.savetxt(f_handle,chi2(noise,data[0],data[1]))

    f_handle.close()

    plt.show()
multiplot(0.1) #run the code for a sigma_src value
```

Appendix F

Code for Getting a Distribution of Multiple Random Fields

```
#! /usr/bin/env python

import math as m
import numpy as n
from matplotlib import pyplot as plt
import pyfits as pf
#####################################################################
#Plotting a histogram of multiple random fields
#####################################################################
def histo(fields,sig,src_sig,noise,sigpow):
  fieldslist = n.array([])
  for x in range(fields): #Opening all random field files and adding all data to a list
    hdu = pf.open('./generations/src_'+str(src_sig)+'/10'+str(sigpow)+'nois'+str(noise)+'/randomfield'+str(x+1)+'.fits')
    data = hdu[0].data
    fieldslist = n.append(fieldslist,data)

  #Determining the root mean square of the entire distribution
  su = 0.
  for a in fieldslist:
    su += a**2.
  su = su/n.shape(fieldslist)[0]
  rms = n.sqrt(su)
  print 'rms = ',rms

  #Making the histogram plot
  plt.figure()
  plt.figtext(0.40, 0.85,r'rms='+str(rms),ha='center', va='center')
  plt.hist(fieldslist,100)
  plt.title(r'Distribution of all '+str(fields)+' random fields together, for $\sigma^2 = 10^{'+str(sigpow)+'}$')
  plt.xlabel('Value')
  plt.ylabel('Number')
  plt.savefig('./generations/src_'+str(src_sig)+'/10'+str(sigpow)+'nois'+str(noise)+'/disttotalranfield.png')
```