Exam Project: Look inside the forest

Valeria Bubuioc s215242 Mireya Torres Uribeetxebarria, s203733
Carolina Hernández Martínez s226440 Jawhara Hamoua, s204746

April 2023

Contents

1 Introduction

The purpose of this report is to help a hardwood floors company design a Computed Tomography (CT) system that can detect and allocate shots in wood logs before cutting, in order to maximise the amount of usable wood.

The underlying mathematical model we created is implemented in Python, and makes use of the functions belonging to the AIRTools package to reconstruct the image after the CT scan. We validated our system through simulation on artificial data, and researched how the system could be improved through a better reconstruction method.

2 Materials and methods

Computed tomography is done by projecting X-ray beams through the sample log, and measuring the attenuation of the beams after they have crossed the sample log. The attenuation of the beams depends on the properties of the materials it encounters. Given that we have enough beams, we can therefore use the resulting measured attenuation to reconstruct an "image" of the material properties inside the log. This reconstruction can be done with the use of a mathematical model derived as follows.

2.1 Mathematical Model

Given an x-ray beam that passes straight through our log sample, we can describe the length the beam has traversed inside the log as ℓ, the materials spatially dependant attenuation coefficient at point ℓ as x, and the intensity of the beam at point ℓ as $I(\ell)$. As explained in Computed Tomography book [1], $I(\ell)$ fulfils Lambert-Beer's law:

$$dI = -xI(\ell)d\ell \tag{1}$$

Through several algebraic derivation steps we reach the equation:

$$I = I_0 e^{-\int_{l=0}^{l_{max}} x d\ell} \tag{2}$$

Given the initial condition $I(0) = I_0$, that ℓ_{max} is the total length the beam has traversed through the sample, and that the material is homogeneous with the attenuation coefficient x_0, we arrive at the usual form of Lambert-Beer's Law:

$$I = I_0 e^{-x_0 \ell_{max}} \tag{3}$$

The full derivation from equation (1) to (2) to (3) can be found in 6 Appendix.

Using equation (2) we can analyse the intensity of the beam all throughout the sample log. If we describe the points that the beam crosses as $j = 1..p$, their attenuation coefficients as $x_j \; with \; j = 1..p$ and the length the beam has crossed at those points as $l_j \; with \; j = 1..p$, we can rewrite equation (2) as:

$$\frac{I_0}{I} = e^{\int_{\ell_j with j=1}^{j=p} x_j d\ell_j} \equiv \sum_{j=1}^{p} x_j \ell_j \tag{4}$$

Taking the logarithm on both sides of equation (4), and assuming that the attenuation coefficient x_j is constant at point j leads to:

$$ln(\frac{I_0}{I}) = \sum_{j=1}^{p} x_j \ell_j \tag{5}$$

We will now call $b = ln(\frac{I_0}{I})$, which represents the data for the given beam. If we discretize the problem and we say that m beams go through the sample we can call the data for a given beam i, $b_i \; with \; i = 1..m$ and the traversed length for a given beam i at point j as $\ell_{ij} \; with \; i = 1..m \; and \; j = 1..p$.

Next we can simplify our notation by saying that all points of the sample which are not hit by beam i are assigned a traversed beam length of 0. This results in the following system of linear equations:

$$b_i = \sum_{j=1}^{p} a_{ij} x_j, \qquad i = 1..m \; and \; a_{ij} = \begin{cases} \ell_{ij} \; if \; beam \; i \; hits \; point \; j \\ 0 \; otherwise \end{cases} \tag{6}$$

This system of linear equations can also be described in the well-known format of $A \cdot x = b$. In order to solve our mathematical problem, we thus have to solve this system of linear equations.

2.2 Materials

In order to be able to apply equation (6) to our problem, we need to define the geometry of our data. We want to calculate the material properties of a thin slice of the log, so we can afterwards reconstruct the image of the log using equation (6), thus locating the shots in the log. In order to do so, we have assumed that our very thin log slice has a length $\times$ width of $L \times L$. We have then discretized it into a grid of $N \cdot N$ pixels, where each of them has a length of $D = L/N$. If we assume that each pixel is so small that each of them has a constant attenuation coefficient, we can apply equation (6) where each point j represents a pixel. Once this discretization is completed the matrix A and b can be obtained to reconstruct the image. The shape and content of matrix A can be seen in 6 Appendix.

2.3 Shot detection and differentiation

Our objective is to identify the pellets with the highest level of efficiency. We are thus interested in determining the minimal value of N that would allow us to detect even the smallest pellets of 2 mm size. The provided image has a grid of 5000 $\times$ 5000 pixels, and is meant to simulate a log diameter of 50 cm, thus resulting in a pixel size of 0.1 mm. This establishes the following relationship between N (the number of pixels) and physical pixel size:

$$\frac{log\ diameter}{N} = physical\ pixel\ size \tag{7}$$

Since the log diameter is at maximum 500 mm and we know what our desired physical size is (2 mm), we can formulate an easy equation to find the minimum required N to detect 2 mm pellets:

$$\frac{500\ mm}{N} \leq 2\ mm \tag{8}$$

Therefore, $N \geq 250$.
Noteworthy is the fact that using $N = 250$ makes the pellet be on the edge of being detectable due to limited spatial resolution. Although detection is theoretically possible, the pellet occupies a single pixel, therefore has reduced visibility. Moreover, having a low spatial resolution makes the images more sensitive to noise, which could lead to inaccurate detection even when just one pixel is affected by noise.

Ideally, in addition to detecting pellets, we would also like to be able to distinguish between steel and lead/bismuth pellets. This of course relies on the material properties. N, which is the grid size, can only affect this indirectly. Specifically, as N decreases, the spatial resolution of the image decreases, making it more difficult to identify small objects, such as the 2 mm steel and lead/bismuth pellets, and differentiate between them based on spatial information alone. Additionally, as N decreases, less pixels represent each pellet, thus leading to a lower signal-to-noise ratio (SNR). Therefore, distinguishing between pellets can become increasingly challenging because the difference between signal (the pellets) and noise becomes less obvious. On top of that, when the spatial resolution is reduced (lower N value), the ability to differentiate between materials based on their physical properties like attenuation coefficients becomes more challenging. This is because limited spatial resolution makes it hard to accurately distinguish the boundaries of objects based on their material properties and precisely define their boundaries. All in all, it is not feasible to assign a specific numerical value to N for successfully differentiating between steel and lead/bismuth pellets; however, it is clear that maintaining a higher spatial resolution (larger N) would improve our chances.

Another thing that relates to the effectiveness of our model is the energy used for x-ray. When choosing an optimal energy value, we first considered that lower energy levels provide better contrast, meaning they effectively distinguish between various structures within the sample, but limited depth penetration, whereas higher energy levels yield deeper penetration at the expense of reduced contrast and increased operation cost. In order to identify the lowest possible energy that ensures thorough sample penetration while maintaining optimal contrast we experimented with the mass energy-transfer coefficient equation for homogeneous mixtures and compounds found in [2] (it was assumed that beech wood is homogeneous):

$$\frac{\mu}{p} = \Sigma w_i (\frac{\mu}{p})_i \tag{9}$$

Using this equation, we calculated the attenuation coefficient of beech, which we found to be a common tree in Danish forests used in floor manufacturing. The attenuation coefficient was calculated at an energy of 200 keV. With the use of equation (3) we were thus able to calculate the remaining beam intensity when the beam had traversed the whole log (assuming a maximum log diameter of 50 cm). The intensity

of the beam was found to be $0.145\% \cdot I_0$. The full calculation of the attenuation coefficient and intensity can be found in the 6Appendix.

By looking at equation (9), it can be seen that at lower beam energies, the attenuation coefficient would increase and thus the beam wouldn't penetrate as deeply into the log, due to more energy being absorbed by the log. Therefore, operating the CT scan at a lower energy levels would be counterproductive, as they would not achieve full sample penetration, which is essential for our analysis.

In conclusion, although the 200 keV option incurs greater costs, it is the only viable choice in this situation.

2.4 Implementation and improvement of the model

We initially implemented the model with the use of the AIRTools function *Paralleltomo* in Python. Just as the problem we were trying to mathematically model, this function creates a two-dimensional tomography test problem with the use of parallel beams at different angles. We used the numpy function *.resize((N,N), img.LANCZOS)* to resize the provided image, in order to work with different numbers of N, that we then used with *Paralleltomo*. It is to be noted that we used the Lanczos resampling algorithm, as it reduces the size of an image averaging the pixel values in a manner that reduces the loss of detail and sharpness in the image.

Using the coefficient matrix A returned by the *Paralleltomo* function, we performed the reconstruction of the image by computing the dot product of A with the output vector b of the linear system.

It is important to note that in order to work with the image provided, we had to flatten the matrix of values out using the NumPy function *.flatten()*.

When performing computed tomography in the real world, there are two main sources of noise introduced: the quantum noise properties of x-ray photons, and the electronic noise originating from the detector system [3]. In order to make the mathematical model more robust in a realistic setting, we thus introduced some noise into the images.
We decided to use Gaussian noise in our image. This is a form of noise that has a smooth, continuous distribution of noise values, so we hoped that it wouldn't disrupt the image to a large extent. This allowed us to test our model in an even more realistic setting and to further test its robustness. It would have been ideal to also test the model when adding Poisson noise, given that the real existing quantum noise is random. Due to the time-limitation of the project, and the high computation time of the model, this was unfortunately not possible to be done, but could be considered in future implementations.

For the purpose of improving the model's implementation, we explored the AIRTools package in depth. We started by applying the *rzr* AIRTools function to our model. Its purpose is to 'clean' a discretized tomography problem [4] by removing zero rows of the coefficient matrix A, and the corresponding elements of matrix b. Zero values don't contribute to the reconstruction of the image, thus the computation time would be reduced as well as its sensibility to noise.

Furthermore, we decided to implement the AIRTools function *kaczmarz*. This function implements Kaczmarz's iterative method for solving a linear system of the form $Ax = b$ (as the ones we have in our mathematical model). This type of reconstruction is considered an Algebraic Reconstruction Technique (ART). The advantages provided by this reconstruction method are many. Firstly, it is computationally efficient as it updates the reconstructed image pixel by pixel, secondly, it is fitting for reconstructing images containing noise, as it only updates the image one measurement at a time. Lastly, it is a highly flexible method, as the algorithm can be adapt and create good reconstructions even with limited data (limited angles for example) [5]. More information regarding this function is provided in the 6 Appendix.

Additionally, we tested our model using different number of angles and rays in each angle, in order to see if we could still get acceptable reconstructions, and how those changes would impact the final reconstruction of the image.
We evaluated whether the reconstruction was acceptable by visualizing it with different levels of noise and considering the condition number. The condition number can be described as the maximum error magnification factor taken over all changes in input, which is the relative forward error (error in the output) divided by the relative backward error (error in the input) (See 6 Appendix). To compare the sensibility to noise, we focused on the relative backward and forward errors and their changes through different noise levels, looking for a low variability, since a larger increase in the relative forward or backward error for the noisy case suggests a higher sensitivity and potentially a poorer reconstructed solution. The model was tested on the numpy array provided as the test image, as well as artificial data which was created by making modifications to the image provided such as adding pellets of the same size in different parts of the log.

3 Experiments and results

As stated in the Section 2.4, we first performed the reconstruction by solving the linear equation system $A \cdot x = b$ using *np.linalg.lstsq*. We ran the model with a selected range of parameters considering the costs of a high number of angles, rays, and the time of computation using a high N, whilst still looking for acceptable reconstructions with noise.
This reconstruction method resulted in reconstructions with a high sensitivity to noise. The results can be seen in the code in 6 Appendix.

From here onwards, due to the high computation time that increasing N caused, we tried to detect a 4 mm pellet, as the one that appeared in the test image. This means that the minimum N needed to detect a 4 mm pellet was 125 instead of 250, as described in Section 2.3.

We then ran our improved model that uses the *rzr* and *kaczmarz* functions, with a range of N from 50 to 200, 60 to 100 angles, and 75 to 100 rays, comparing with different levels of Gaussian noise. These results can be seen in Figure1, and more in-depth and with a bigger resolution in 6 Appendix.

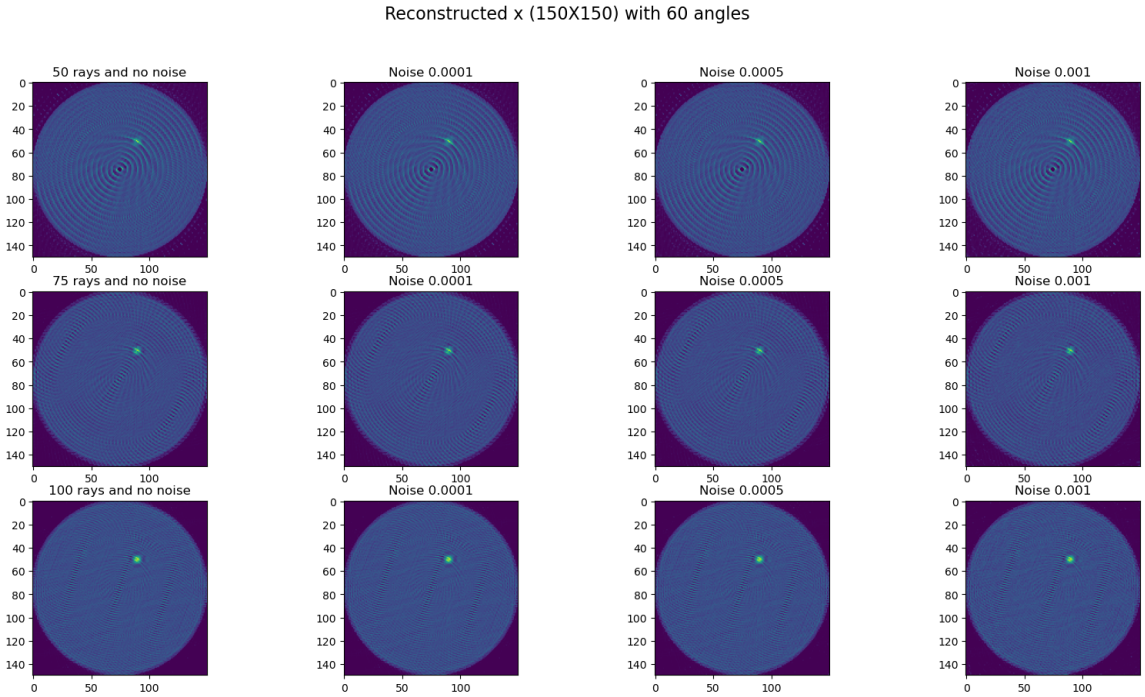


Figure 1: Reconstructions 150x150, 60 angles

We also plotted the relative backward and forward errors as shown in the graphs in Figure 3. When N was small, the error was also small, but would rapidly increase with the introduction of noise. On the other hand, when N was large, the error was larger but it wasn't as susceptible to changes when noise was introduced. The number of rays also influenced the errors, which maintained a similar level with 75 rays or more.

In order to validate our data we proceeded to run the model using artificial data, created by modifying our original data and adding pellets of the same size to the log. For this test-run we kept the range of N from 75 to 125, 60 to 90 angles, and 75 to 100 rays. The results can be seen in Figure 4a.

Moreover, we computed once again the relative error of the results on the artificial data. After analyzing our relative error plot results (Figure 4b) through the different levels of noise in the artificial data, we can see how stable the error is maintained at a 125 x 125 resample size. This resolution was not so heavy on the computing and gave out good reconstructions as seen in figure 4a.

4 Discussion

Throughout our experiments and investigations we have seen that the bigger the N better of a reconstruction we get with and without noise. However, the computing time also increases significantly, and at times, the memory of our Python host wouldn't even finish computing since the memory was surpassed. Considering price, computing, and results, the use of the AIRTools functions aids in a significant way in terms of computing time and reconstruction. However, there needs to be a big consideration of the other

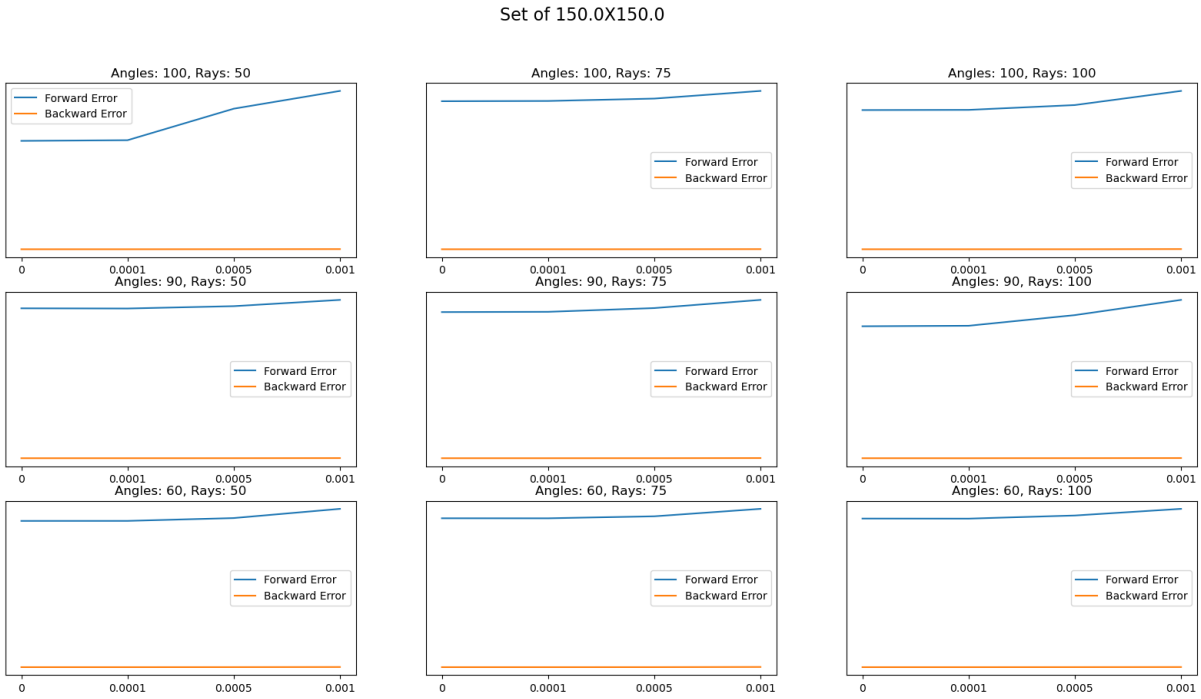
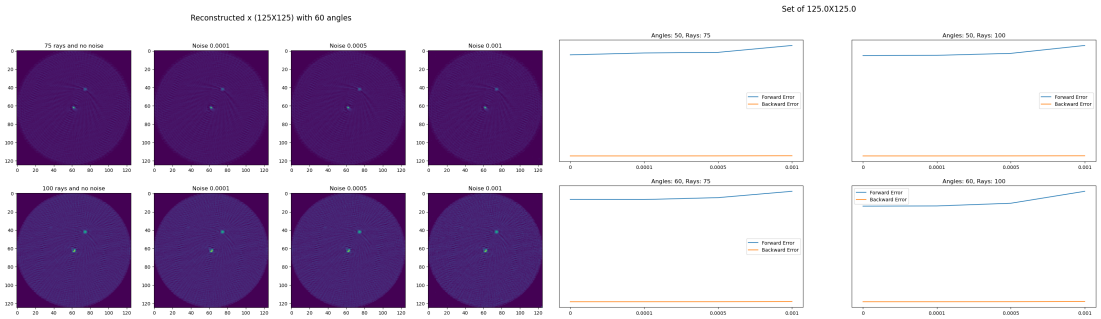


Figure 2: 150x150

Figure 3: Relative errors plotted with different noise levels



(a) Reconstructions 125x125, 60 angles

(b) Relative Errors through noises

Figure 4: Reconstruction 150x150, 60 angles and relative errors plotted with different noise levels, with artificial data

parameters being tested, as larger N values require more angles and rays for acceptable reconstructions. The other parameters we tested such as increasing the number of angles, increased significantly the computing time since it had to compute a much bigger matrix A, but it also meant a better reconstruction and less sensitivity to noise. Moreover, increasing the number of rays per angle also resulted in a better reconstruction regardless of the number of angles since most of the errors seemed to stabilize when at least 100 rays were being used.

5 Conclusion

After thorough research and experimentation, given the scenario where we have pellets of 4 mm, and a log size of 0.5 m, we have found that for acceptable reconstructions, our model can use a value of N between 100 and 200, and around 60 to 100 angles with 75 to 100 rays each. However, it is worth noting that computed tomography and the reconstruction of images depend largely on the context.
Further improvements to the model could be made such as using a computer with a higher computational power which could run a higher value for N, and the addition of Poisson noise.

The mathematical model presented in this report holds considerable industrial value and relevance, as effective detection and differentiation of pellets would enable floor manufacturing companies to design log cuts that maximize the use of wood and minimize waste in future operations.

References

[1] P. C. Hansen & J. S. Jørgensen a& W. R. B. Lionheart (Eds.) *Computed Tomography: Algorithms, Insight, and Just Enough Theory*. 2021.

[2] NIST. *X-Ray Mass Attenuation Coefficients*. URL: https://physics.nist.gov/PhysRefData/XrayMassCoef/chap3.html.

[3] EN Manson & V Atuwo Ampoh & E Fiagbedzi & J H Amuasi & J J Flether & C Schandorf. *Image Noise in Radiography and Tomography: Causes, Effects and Reduction Techniques*. 2019.

[4] Michael Hirsch. *AIRTools*. Oct. 2019. URL: https://pypi.org/project/airtools/.

[5] RICHARD C. ASTER & BRIAN BORCHERS & CLIFFORD H. THURBER. *Parameter Estimation and Inverse Problems*. Elsevier, 2019.

[6] Supported by the European Social Fund and the state budget of the Czech Republic. *Elemental composition of beech wood*. URL: https://akela.mendelu.cz/~xcepl/inobio/nove/Wood_anatomy/WAEF-02-chemical_composition.pdf.

[7] NIST. *X-Ray Mass Attenuation Coefficients by elements*. URL: https://physics.nist.gov/PhysRefData/XrayMassCoef/tab3.html.

6 Appendix

6.1 Mathematical modeling

6.1.1 Beer-Lambert law equation derivation

Beer-Lambert law equation derivation Given the following variables:

- ℓ length the beam has traversed inside the log
- x the materials spatially dependant attenuation coefficient ($x(\ell)$ would describe the attenuation coefficient at point ℓ)
- $I(\ell)$ the intensity of the beam at point ℓ
- $I(0) = I_0$ the intensity of the beam at $\ell = 0$ (initial intensity)
- ℓ_{max} the total length the beam has traversed through the sample

We start with the differential equation:

$$dI = -xI(\ell)d\ell \tag{10}$$

Which can be re-organised into:

$$\frac{1}{I(\ell)}dI = -xd\ell \tag{11}$$

We then compute the integral over both sides of the equation and we obtain:

$$\int_{I=0}^{I} \frac{1}{I(\ell)}dI = -\int_{l=0}^{l_{max}} xd\ell \tag{12}$$

$$\equiv ln(\frac{I}{I_0}) = -\int_{l=0}^{l_{max}} xd\ell \tag{13}$$

If we take the exponential on both sides of equation (13) we get:

$$\frac{I}{I_0} = e^{-\int_{l=0}^{l_{max}} xd\ell} \tag{14}$$

If we were to assume that the material is homogeneous with a linear attenuation coefficient x_0 we can approximate the integral on the right hand side of equation (14) by:

$$\int_0^{\ell_{max}} xd\ell \approx x_0 \int_0^{\ell_{max}} d\ell = x_0\ell_{max} \tag{15}$$

We thus get the following equation:

$$\frac{I}{I_0} = e^{-x_0\ell_{max}} \tag{16}$$

Finally if we multiply by I_0 we get the Lamber Beer Law equation described above under the Mathematical model section:

$$I = I_0e^{-x_0\ell_{max}} \tag{17}$$

6.1.2 Matrix A

We assume that the CCD camera is positioned along the log slice's two edges yielding just two projections. We assume that each pixel is precisely hit by one X-ray, with rays parallel to the the s-axis or the t-axis and passing through exactly N pixels, as shown in Figure 5.

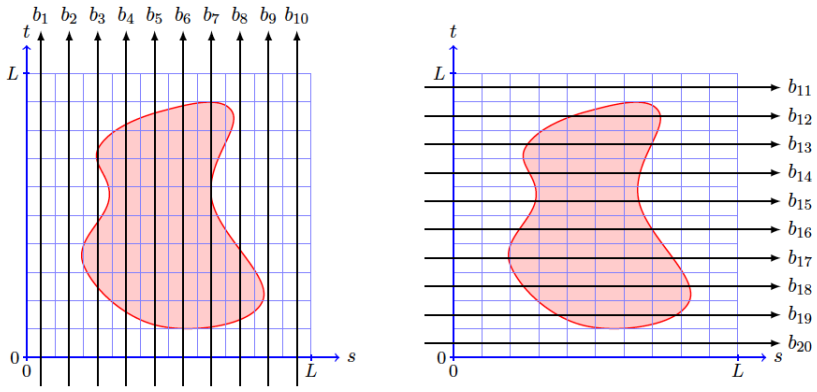


Figure 5: Configuration with $N = 10$ and two projections.

Using this information and the knowledge we have about the linear system of equations found in equation (6), we can further describe the shape and content of matrix A.
We know that matrix A is rectangular with $m = 2N$ rows and $n = N^2$ columns. Each of the rows in the matrix represent the lengths of the pixels that beam i has hit. That is why in total we have $2N$ rows, which is the same as the total number of beams. We also know that the total number of pixels that beam i goes across is N, and thus in every row of A we will have N lengths, and the rest of the values in the row $(N^2 - N)$ will be 0.
In other words, if we were to say that we have $N = 3$, with a grid of 3×3 pixels, we can describe A as follows. The first row of matrix A would represent the first ray that crosses parallel to the t-axis, across the first 'column' of pixels (b_1 in 5), and would have values:

$$[\Delta L, \Delta L, \Delta L, 0, 0, 0, 0, 0, 0]$$

The second row would represent the second ray and would look like:

$$[0, 0, 0, \Delta L, \Delta L, \Delta L, 0, 0, 0]$$

Overall, we would have the following A:

$$\begin{bmatrix} \Delta L, \Delta L, \Delta L, 0, 0, 0, 0, 0, 0 \\ 0, 0, 0, \Delta L, \Delta L, \Delta L, 0, 0, 0 \\ 0, 0, 0, 0, 0, 0, \Delta L, \Delta L, \Delta L \\ \Delta L, 0, 0, \Delta L, 0, 0, \Delta L, 0, 0 \\ 0, \Delta L, 0, 0, \Delta L, 0, 0, \Delta L, 0 \\ 0, 0, \Delta L, 0, 0, \Delta L, 0, 0, \Delta L \end{bmatrix}$$

In this example we used a very low value for N, but the concept for the shape and content of matrix A expands to larger values of N.

6.1.3 Kaczmarz's algorithm

Kaczmarz's algorithm is an 'Algebraic Reconstruction Technique', originally proposed in 1937, and independently suggested under the name ART by Gordon, Bender, Herman in 1970 for tomographic reconstruction.

Kaczmarz's algorithm first makes an initial estimate of the image that is then updated in an iterative manner until the projection is matched as close as possible. This means that the algorithm updates one pixel at a time in the image, based on the projection data and the geometry of the system.

6.2 Shot detection and differentiation

The following mass energy-transfer coefficient equation for homogeneous mixtures and compounds was used:

$$\frac{\mu}{p} = \Sigma w_i (\frac{\mu}{p})_i \tag{18}$$

where $\frac{\mu}{p}$ denotes the mass attenuation coefficient of the compound, while w_i is the weight percentage and $(\frac{\mu}{p})_i$ is the mass coefficient of each element in the compound.

The composition of beech consists mainly of C, H, O, N. Inserting their weight percentages [6] and individual mass coefficients at 200 keV [7] leads to the following equation:

$$0.497 \cdot 1.229 \cdot 10^{-1} \frac{cm^2}{g} + 0.062 \cdot 2.429 \cdot 10^{-1} \frac{cm^2}{g} + 0.436 \cdot 1.237 \cdot 10^{-1} \frac{cm^2}{g} + 0.005 \cdot 1.233 \cdot 10^{-1} \frac{cm^2}{g} = 0.1306908 \frac{cm^2}{g} \tag{19}$$

The remaining intensity I was calculated as follows:

$$I = I_0 e^{-x_0 l_{max}} \tag{20}$$

Assuming I_0 is 100%,

$$I = 100\% \cdot e^{-0.1306908 \frac{cm^2}{g} \cdot 50cm} = 0.1452396977\% \tag{21}$$

This resulted in a remaining intensity of 0.145%, indicating that the X-ray barely penetrated the entire sample.

6.3 Condition number

Condition number:

$$\|A\| \left\|A^{-1}\right\| \tag{22}$$

Backward error:

$$\frac{\|b - A\hat{x}\|}{\|b\|} \tag{23}$$

Forward error:

$$\frac{\|x - \hat{x}\|}{\|x\|} \tag{24}$$

6.4 Python code

Exam Project

Downsampling image size

In [1]:
```
from PIL import Image
import os
import paralleltomo #We had to change float to float32


import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

```
C:\Users\Owner\Desktop\exam_project\paralleltomo.py:102: RuntimeWarning: divide by ze
ro encountered in true_divide
  tx = (x - x0theta[j,0])/a
```

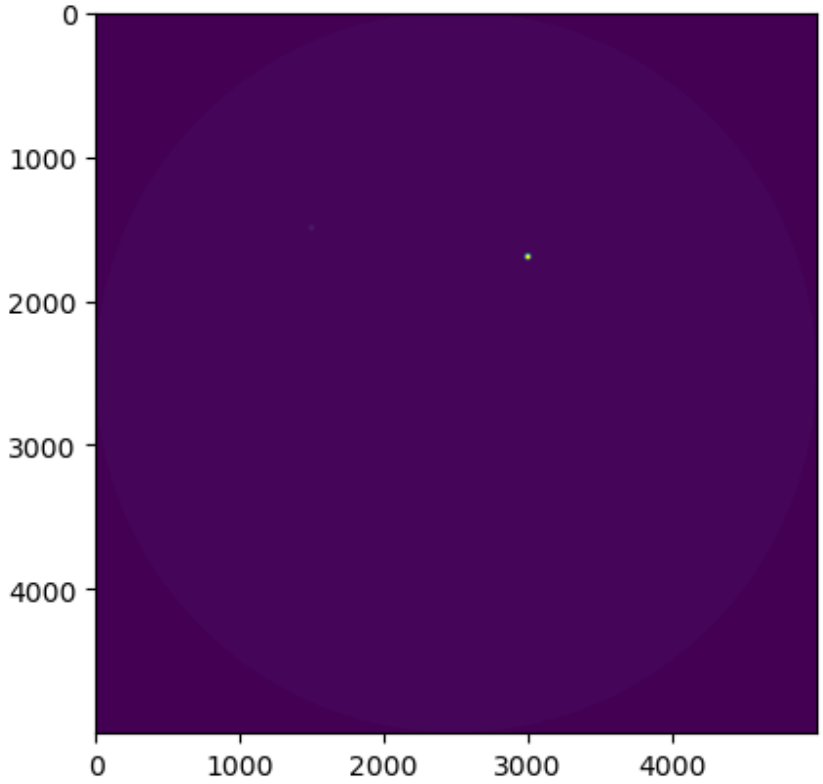
Load image and get values from array

In [49]:
```
import numpy as np

# Load the NumPy array from the file
array_data = np.load('testImage.npy')

# Get the unique values and their counts
unique_values, value_counts = np.unique(array_data, return_counts=True)

# Print the unique values and their counts
for value, count in zip(unique_values, value_counts):
    print(f"Value: {value}, Count: {count}")
```

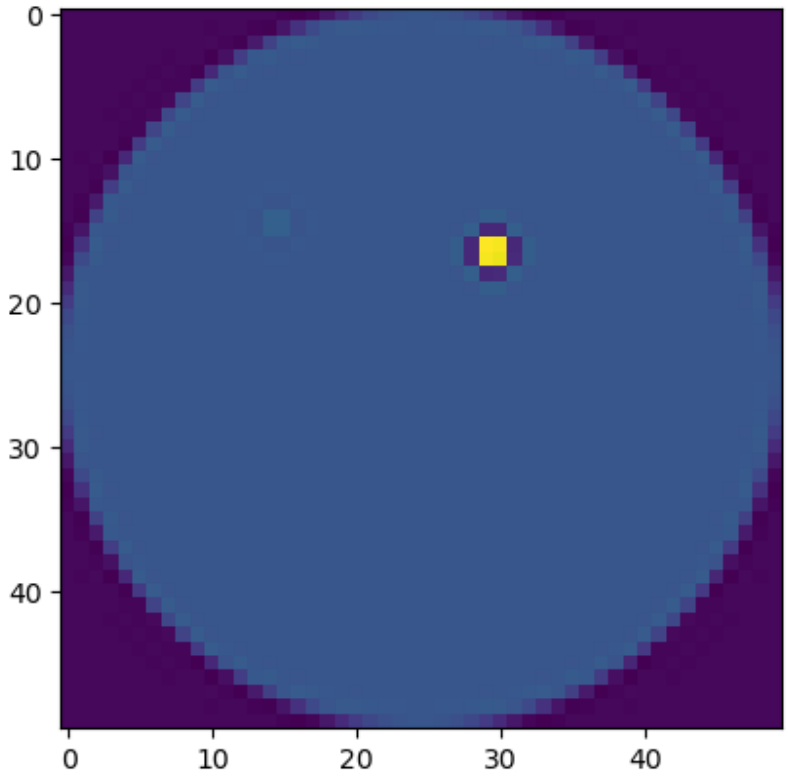
```
Value: 0.0, Count: 5365197
Value: 0.0008496388523324683, Count: 19632289
Value: 0.0037101005066195736, Count: 1257
Value: 0.05397851616202787, Count: 1257
```

Plot the original image and the resized image

In [101…
```
# Load image and convert it to a PIL Image object
img_array = np.load("testImage.npy")
img = Image.fromarray(img_array)
# show the image using matplotlib
plt.imshow(img_array)
plt.show()

# Resize the image from 5000x5000 pixels to 50x50 pixels
img = img.resize((50,50), Image.LANCZOS)

# Convert image again from PIL to numpy
img_array = np.array(img)

print(img_array.shape)
plt.imshow(img_array)
plt.show()
```




```
C:\Users\Owner\AppData\Local\Temp\ipykernel_13468\4230183708.py:9: DeprecationWarnin
g: LANCZOS is deprecated and will be removed in Pillow 10 (2023-07-01). Use Resamplin
g.LANCZOS instead.
  img = img.resize((50,50), Image.LANCZOS)
(50, 50)
```



Paralleltomo reconstruction

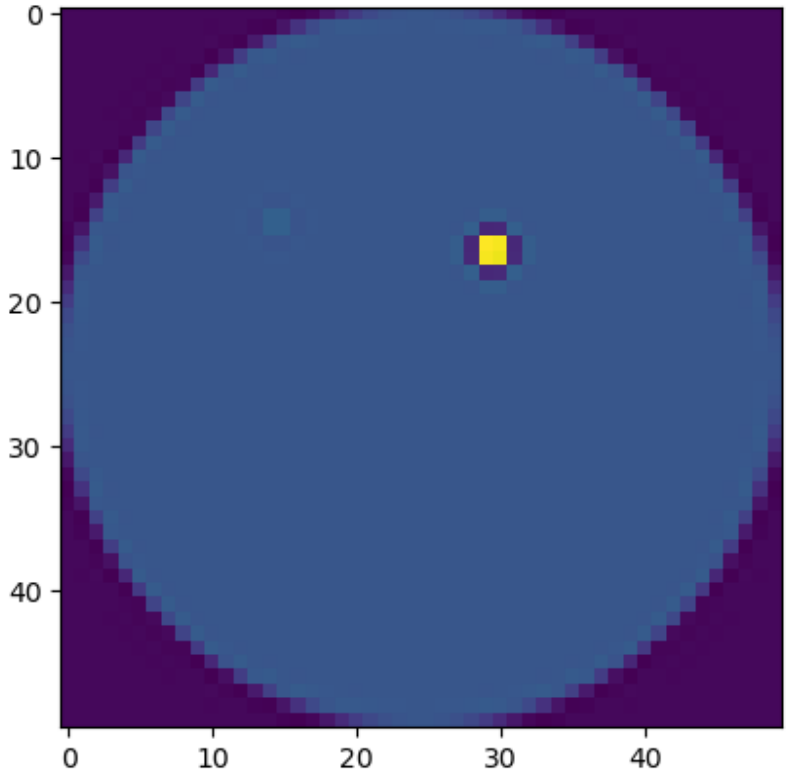
In [102…
```
import numpy as np
#N=50/np.sqrt(2)
#[A,theta,p,d] = paralleltomo.paralleltomo(N)
N=50
theta =np.matrix(range(0,180,3)) #range creates a sequence of numbers that starts at 0
p=100

[A,theta,p,d] = paralleltomo.paralleltomo(N, theta, p)

rank = np.linalg.matrix_rank(A)
print(rank)
print(A.shape)
```

```
2500
(6000, 2500)
```

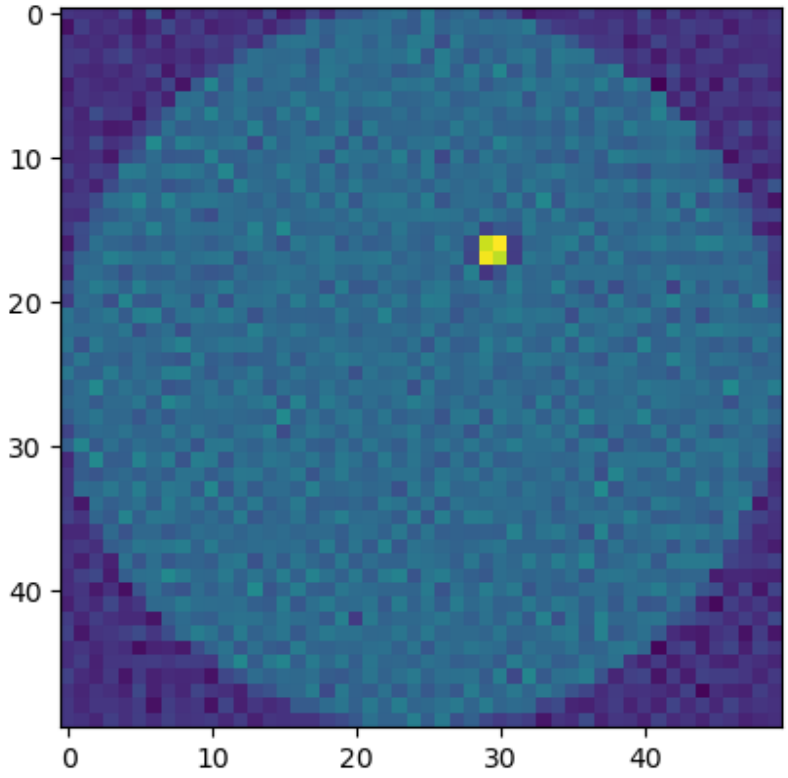
In [103…
```
import matplotlib.pyplot as plt

x=img_array.flatten()
b=np.dot(A,x)

x_reconstructed, residuals, rank, s = np.linalg.lstsq(A,b, rcond=None)
print(x_reconstructed)
```

```
[-3.65257918e-17  4.16333634e-17 -1.28369537e-16 ...  6.50521303e-19
 -1.08420217e-18 -1.08420217e-18]
```

In [107…
```
img_array = np.load("testImage.npy")
img = Image.fromarray(img_array)
# calculate the original shape of the image
orig_shape = (50,50)
# reshape the flattened image to the original shape
recovered_img_x = x_reconstructed.reshape(orig_shape)
# convert the image back to PIL Image format
recovered_img_x = np.array(recovered_img_x)
plt.imshow(recovered_img_x )
plt.show()


#ADD NOISE
noise= np.random.normal(0,0.001, b.shape)
b_noise= b + noise
x_noise, residuals_noise, rank_noise, s = np.linalg.lstsq(A,b_noise, rcond=None)
print(x_noise)

# reshape the flattened image to the original shape
recovered_img_x_noise = x_noise.reshape(orig_shape)
# convert the image back to PIL Image format
recovered_img_x_noise = np.array(recovered_img_x_noise)
plt.imshow(recovered_img_x_noise)
plt.show()
```




```
[ 6.48611963e-05  4.29587464e-06 -1.78086842e-06 ... -1.72093070e-04
 -9.07465079e-05 -3.45504606e-05]
```



Try out different settings of Paralleltomo (angles, number of rays, etc.), different noise levels, and different resolutions of the test image.

Different scenarios using Paralleltomo method just different resample sizes, number of angles, and number of rays

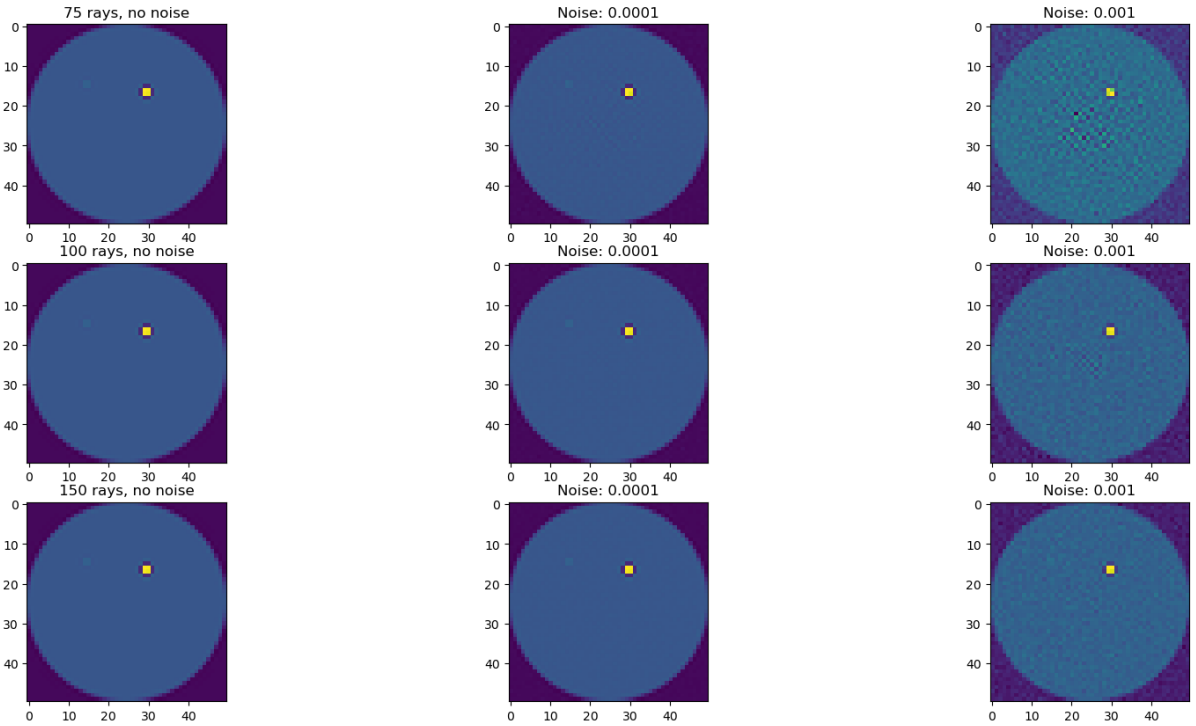
In [8]:
```
# Load the image with lead and steel shot
import matplotlib.pyplot as plt


# Load image and convert it to a PIL Image object
img_array = np.load("testImage.npy")
img = Image.fromarray(img_array)

#Parameters to resample x, the values divided by 2 give N, to form an NXN grid
# for example, first value 50 will give a 25x25 grid
#modify and/or add as you want, just note that the bigger the grid the more it will la

resample_x = [100,200]

test = []


for i in resample_x:

    L = int(i/2)
    # Resize the image from 5000x5000 pixels to 50x50 pixels
    img = img.resize((L, L), Image.Resampling.LANCZOS)

    # Convert image again from PIL to numpy
    img_array = np.array(img)
    x=img_array.flatten()
    N = img_array.shape[0]
    print(f'Grid of {L} X {L}')

    #You can modify and/or add more variations for the next parameters


    #These are the number of angles that we may use, we will later on use np.linspace(
    # values between 0 and 180

    angles = [100,50]

    #number of rays for each angle
    rays = [75,100,150]

    #amount of noise
    noises = [1e-4, 1e-3]

    #ignore
    pt=0

    test1 = []

    for j in angles:

        #create figure for subplots for each number of angles
```

```
fig, axs = plt.subplots(len(rays),len(noises)+1, figsize=(20, 10), facecolor=
#fig.subplots_adjust()
fig.suptitle(f'Reconstructed x ({L}X{L}) with {j} angles', fontsize=20)
axs = axs.ravel()
ax = 0


for n in rays:

    #set parameters for paralleltomo

    theta_i = np.matrix(np.linspace(0,180,j))
    p=n

    #paralleltomo
    [A,thetar,pr,d] = paralleltomo.paralleltomo(N, theta_i, p)

    #obtain b
    b=np.dot(A,x)

    x_reconstructed, residuals, rank, s = np.linalg.lstsq(A,b, rcond = None)

    orig_shape = (N, N)
    # reshape the flattened image to the original shape
    recovered_img_x = x_reconstructed.reshape(orig_shape)
    # convert the image back to PIL Image format
    recovered_img_x = np.array(recovered_img_x)

    cond = np.linalg.cond(A, 2)
    #print(f'condition number: {cond}')

    r_v = (A@x_reconstructed)-b
    A_inv = np.linalg.pinv(A)
    deltax_norm= (np.linalg.norm(-A_inv@r_v))
    rel_res = (np.linalg.norm(r_v))/((np.linalg.norm(A))*(np.linalg.norm(x)))
    rel_error = (np.linalg.norm(deltax_norm))/(np.linalg.norm(x))


    axs[ax].imshow(recovered_img_x )
    #txt = str(f'CN of A: {cond}'
              #f',relative error/residuals:'
              #f' {rel_error}, {rel_res}')
    #axs[ax].text(.5, .05, txt, ha='right', va='bottom')
    axs[ax].set_title(f'{p} rays, no noise')

    test1.append([cond,rel_res, rel_error])


    #print('it gives a condition number greater or equal than:')
    #print(leh/rih)

    ax+=1
    for a in noises:

      #ADD NOISE
        noise=np.random.normal(0,a, b.shape)
        b_noise= b + noise
        x_noise, residuals, rank, s = np.linalg.lstsq(A,b_noise, rcond=None)

        # reshape the flattened image to the original shape
```

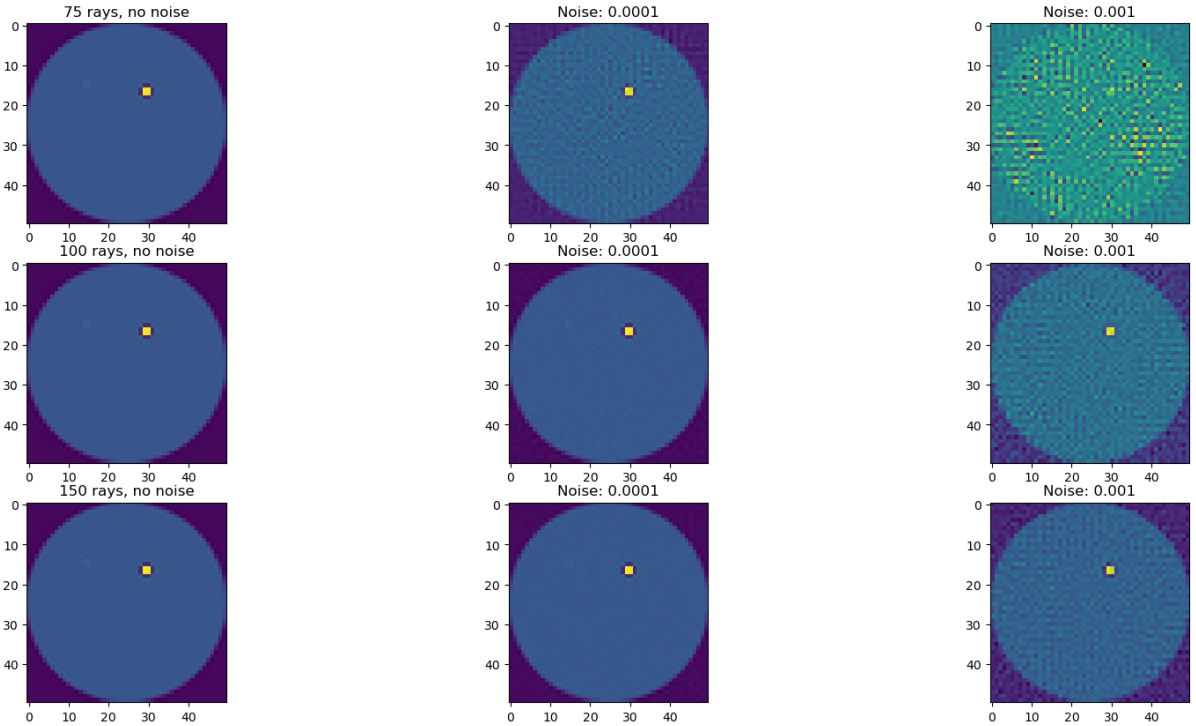
```
                recovered_img_x_noise = x_noise.reshape(orig_shape)
                # convert the image back to PIL Image format
                recovered_img_x_noise = np.array(recovered_img_x_noise)

                cond = np.linalg.cond(A, 2)
                #print(f'condition number: {cond}')

                r_v = (A@x_noise)-b
                A_inv = np.linalg.pinv(A)
                deltax_norm= (np.linalg.norm(-A_inv@r_v))
                rel_res = (np.linalg.norm(r_v))/((np.linalg.norm(A))*(np.linalg.norm(x
                rel_error = (np.linalg.norm(deltax_norm))/(np.linalg.norm(x))
                EMF = rel_error/rel_res

                test1.append([EMF, rel_res, rel_error])
                axs[ax].imshow(recovered_img_x_noise)
                #txt = str(f'CN =>:{cond}'
                          #f'Relative error/residuals='
                          #f' {rel_error}, {rel_res}')
                #axs[ax].text(.5, .05, txt, ha='right', va='bottom')
                axs[ax].set_title(f'Noise: {a}')
                ax+=1

        pt+=1
        print(f'{pt} / {len(angles)}')
        #pt+=1

    test.append([test1])
```

```
Grid of 50 X 50
1 / 2
2 / 2
Grid of 100 X 100
1 / 2
2 / 2
```

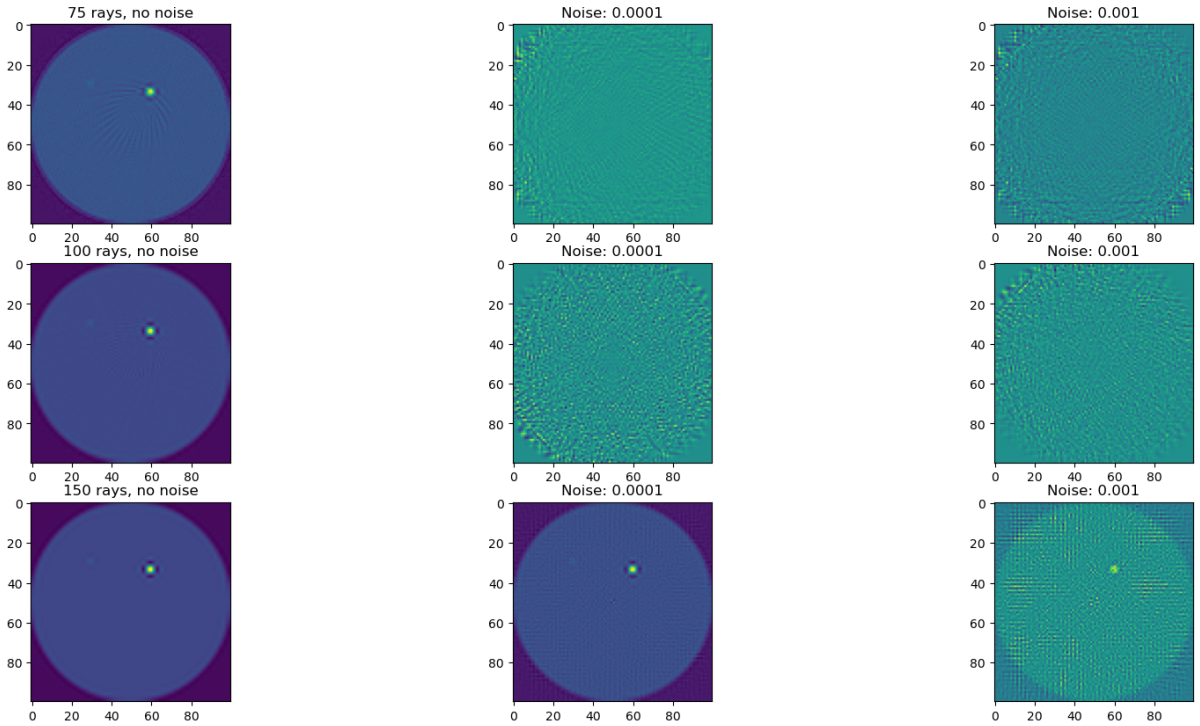
Reconstructed x (50X50) with 100 angles



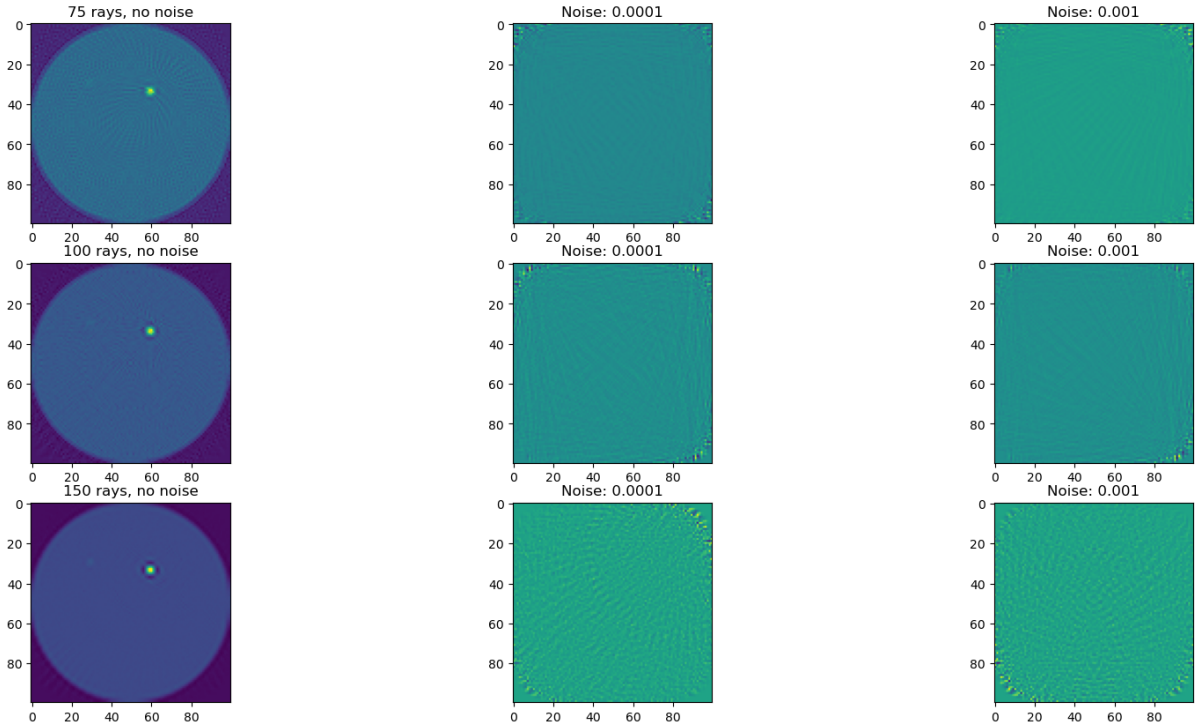
Reconstructed x (50X50) with 50 angles



Reconstructed x (100X100) with 100 angles



Reconstructed x (100X100) with 50 angles



Using airtools rzr() to optimize matrix A helping it be less sensible to noise, and kaczmarz() for a better x reconstruction

In [2]:
```
# Load the image with lead and steel shot
import matplotlib.pyplot as plt
import airtools



# Load image and convert it to a PIL Image object
img_array = np.load("testImage.npy")
img = Image.fromarray(img_array)

#Parameters to resample x, the values divided by 2 give N, to form an NXN grid
# for example, first value 50 will give a 25x25 grid
#modify and/or add as you want, just note that the bigger the grid the more it will la

resample_x = [100,200,300,400]

testA = []

for i in resample_x:

    L = int(i/2)
    # Resize the image from 5000x5000 pixels to 50x50 pixels
    img = img.resize((L, L), Image.Resampling.LANCZOS)

    # Convert image again from PIL to numpy
    img_array = np.array(img)
    x=img_array.flatten()
    N = img_array.shape[0]
    print(f'Grid of {L} X {L}')

    #You can modify and/or add more variations for the next parameters


    #These are the number of angles that we may use, we will later on use np.linspace(
    # values between 0 and 180

    angles = [100,90,60]

    #number of rays for each angle
    rays = [50,75,100]

    #amount of noise
    noises = [1e-4, 5e-4, 1e-3]

    #ignore
    pt=0
    testA1 = []

    for j in angles:

        #create figure for subplots for each number of angles
        fig, axs = plt.subplots(len(rays),len(noises)+1, figsize=(20, 10), facecolor=
        fig.subplots_adjust(hspace = .2)
        fig.suptitle(f'Reconstructed x ({L}X{L}) with {j} angles', fontsize=16)
        axs = axs.ravel()
        ax = 0

        for n in rays:
```

```
            #set parameters for paralleltomo

            theta_i = np.matrix(np.linspace(0,180,j))
            p=n

            #paralleltomo
            [A,thetar,pr,d] = paralleltomo.paralleltomo(N, theta_i, p)

            #obtain b
            b=np.dot(A,x)

            A, b, som = airtools.rzr(A,b)
            #condA = np.linalg.cond(A)
            x_reconstructed, residuals = airtools.kaczmarz(A,b)
            condA = np.linalg.cond(A, 2)
            #print(f'condition number: {cond}')

            r_v = (A@x_reconstructed)-b
            A_inv = np.linalg.pinv(A)
            deltax_norm= (np.linalg.norm(-A_inv@r_v))
            rel_res = (np.linalg.norm(r_v))/((np.linalg.norm(A))*(np.linalg.norm(x)))
            rel_error = (np.linalg.norm(deltax_norm))/(np.linalg.norm(x))
            #EMF = rel_error/rel_res
            #print(f'Condition number of A with N = {N}, nA = {j}, p = {p}: {condA}')
            #print(f'Relative error of reconstructed x: {rel_error}')
            #print(f'Relative residual: {rel_res} ')
            testA1.append([condA, rel_res, rel_error])
            #x_reconstructed, residuals = airtools.kaczmarz(A,b)

            orig_shape = (N, N)
            # reshape the flattened image to the original shape
            recovered_img_x = x_reconstructed.reshape(orig_shape)
            # convert the image back to PIL Image format
            recovered_img_x = np.array(recovered_img_x)


            axs[ax].imshow(recovered_img_x )
            axs[ax].set_title(f'{p} rays and no noise')
            #sensitivity = np.linalg.cond(A)


            ax+=1
            for a in noises:

              #ADD NOISE
                noise=np.random.normal(0,a, b.shape)
                b_noise= b + noise
                A, b_noise, som = airtools.rzr(A,b_noise)

                x_noise, residuals = airtools.kaczmarz(A,b_noise)

                r_v = (A@x_noise)-b
                A_inv = np.linalg.pinv(A)
                deltax_norm= (np.linalg.norm(-A_inv@r_v))
                rel_res = (np.linalg.norm(r_v))/((np.linalg.norm(A))*(np.linalg.norm(x
                rel_error = (np.linalg.norm(deltax_norm))/(np.linalg.norm(x))
                EMF = rel_error/rel_res
                #print(f'EMF with noise = {a}: {EMF}')
                #print(f'Relative error of reconstructed x: {rel_error}')
                #print(f'Relative residual: {rel_res} ')
```

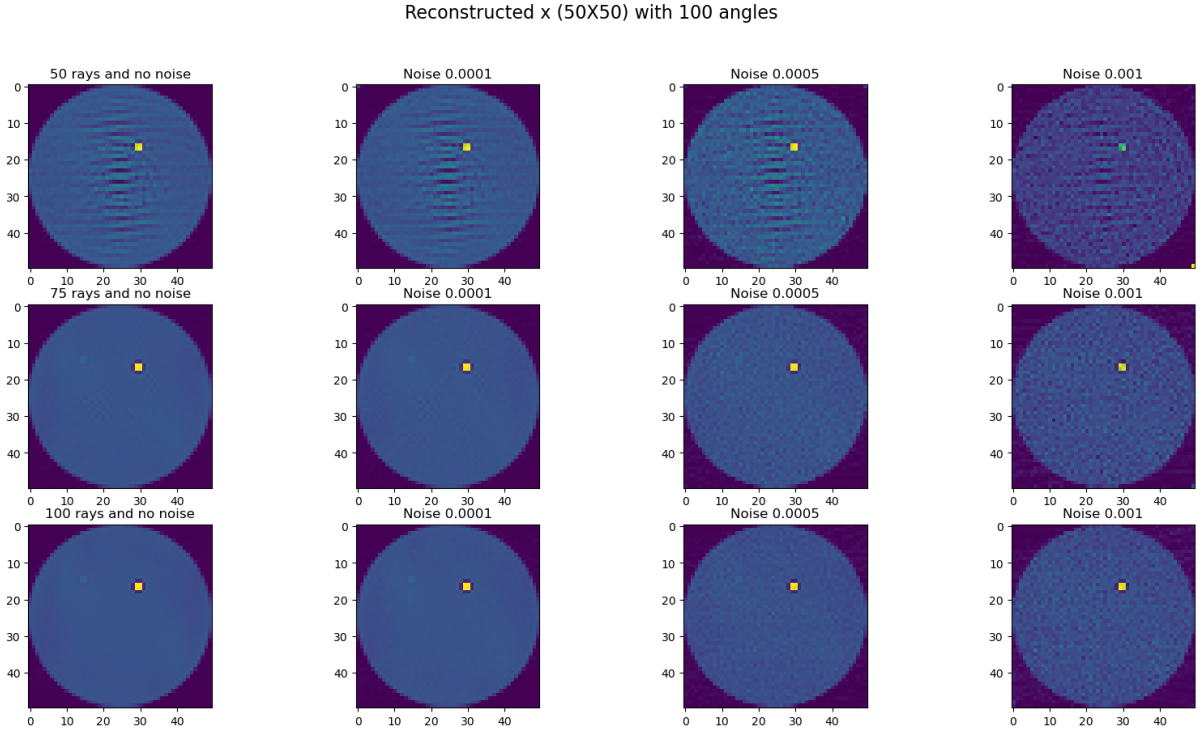
```
                testA1.append([EMF, rel_res, rel_error])

                # reshape the flattened image to the original shape
                recovered_img_x_noise = x_noise.reshape(orig_shape)
                # convert the image back to PIL Image format
                recovered_img_x_noise = np.array(recovered_img_x_noise)

                axs[ax].imshow(recovered_img_x_noise)
                axs[ax].set_title(f'Noise {a}')
                ax+=1
        pt+=1
        print(f'{pt} / {len(angles)}')
        #pt+=1

    testA.append([testA1])
```

```
Grid of 50 X 50
C:\Users\Owner\Desktop\exam_project\parallelto mo.py:102: RuntimeWarning: divide by ze ro encountered in true_divide
  tx = (x - x0theta[j,0])/a
Iteration 0,  ||residual|| = 0.31
Iteration 0,  ||residual|| = 0.31
Iteration 0,  ||residual|| = 0.32
Iteration 0,  ||residual|| = 0.32
C:\Users\Owner\Desktop\exam_project\parallelto mo.py:102: RuntimeWarning: invalid valu e encountered in true_divide
  tx = (x - x0theta[j,0])/a
```

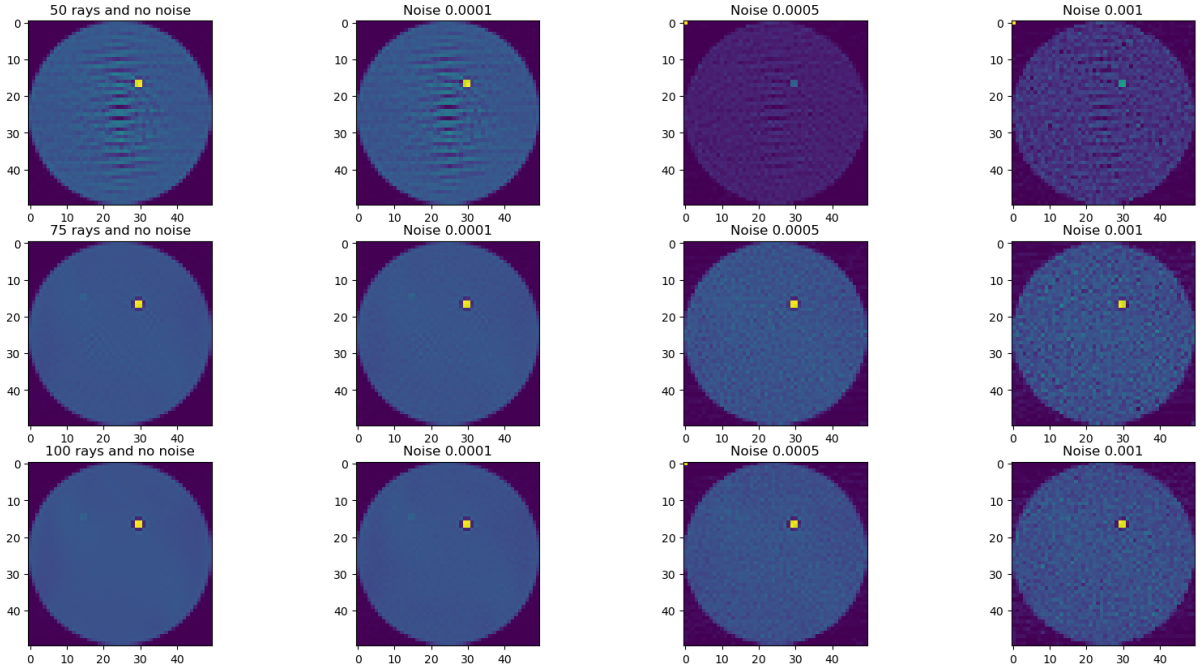
```
Iteration 0,  ||residual|| = 0.15
Iteration 0,  ||residual|| = 0.15
Iteration 0,  ||residual|| = 0.15
Iteration 0,  ||residual|| = 0.16
Iteration 0,  ||residual|| = 0.17
Iteration 0,  ||residual|| = 0.17
Iteration 0,  ||residual|| = 0.17
Iteration 0,  ||residual|| = 0.19
1 / 3
Iteration 0,  ||residual|| = 0.29
Iteration 0,  ||residual|| = 0.29
Iteration 0,  ||residual|| = 0.29
Iteration 0,  ||residual|| = 0.29
Iteration 0,  ||residual|| = 0.14
Iteration 0,  ||residual|| = 0.14
Iteration 0,  ||residual|| = 0.14
Iteration 0,  ||residual|| = 0.15
Iteration 0,  ||residual|| = 0.16
Iteration 0,  ||residual|| = 0.16
Iteration 0,  ||residual|| = 0.17
Iteration 0,  ||residual|| = 0.18
2 / 3
Iteration 0,  ||residual|| = 0.22
Iteration 0,  ||residual|| = 0.22
Iteration 0,  ||residual|| = 0.23
Iteration 0,  ||residual|| = 0.23
Iteration 0,  ||residual|| = 0.11
Iteration 0,  ||residual|| = 0.11
Iteration 0,  ||residual|| = 0.11
Iteration 0,  ||residual|| = 0.12
Iteration 0,  ||residual|| = 0.13
Iteration 0,  ||residual|| = 0.13
Iteration 0,  ||residual|| = 0.14
Iteration 0,  ||residual|| = 0.15
3 / 3
Grid of 100 X 100
Iteration 0,  ||residual|| = 0.71
Iteration 0,  ||residual|| = 0.71
Iteration 0,  ||residual|| = 0.71
Iteration 0,  ||residual|| = 0.71
Iteration 0,  ||residual|| = 0.82
Iteration 0,  ||residual|| = 0.82
Iteration 0,  ||residual|| = 0.82
Iteration 0,  ||residual|| = 0.81
Iteration 0,  ||residual|| = 0.87
Iteration 0,  ||residual|| = 0.87
Iteration 0,  ||residual|| = 0.87
Iteration 0,  ||residual|| = 0.88
1 / 3
Iteration 0,  ||residual|| = 0.65
Iteration 0,  ||residual|| = 0.65
Iteration 0,  ||residual|| = 0.65
Iteration 0,  ||residual|| = 0.65
Iteration 0,  ||residual|| = 0.77
Iteration 0,  ||residual|| = 0.77
Iteration 0,  ||residual|| = 0.77
Iteration 0,  ||residual|| = 0.78
Iteration 0,  ||residual|| = 0.89
Iteration 0,  ||residual|| = 0.89
Iteration 0,  ||residual|| = 0.89
```

```
Iteration 0,  ||residual|| = 0.89
2 / 3
Iteration 0,  ||residual|| = 0.43
Iteration 0,  ||residual|| = 0.43
Iteration 0,  ||residual|| = 0.43
Iteration 0,  ||residual|| = 0.44
Iteration 0,  ||residual|| = 0.74
Iteration 0,  ||residual|| = 0.74
Iteration 0,  ||residual|| = 0.74
Iteration 0,  ||residual|| = 0.74
Iteration 0,  ||residual|| = 0.69
Iteration 0,  ||residual|| = 0.69
Iteration 0,  ||residual|| = 0.69
Iteration 0,  ||residual|| = 0.70
3 / 3
Grid of 150 X 150
Iteration 0,  ||residual|| = 0.89
Iteration 0,  ||residual|| = 0.89
Iteration 0,  ||residual|| = 0.89
Iteration 0,  ||residual|| = 0.89
Iteration 0,  ||residual|| = 1.08
Iteration 0,  ||residual|| = 1.08
Iteration 0,  ||residual|| = 1.08
Iteration 0,  ||residual|| = 1.08
Iteration 0,  ||residual|| = 1.57
Iteration 0,  ||residual|| = 1.57
Iteration 0,  ||residual|| = 1.57
Iteration 0,  ||residual|| = 1.56
1 / 3
Iteration 0,  ||residual|| = 0.83
Iteration 0,  ||residual|| = 0.83
Iteration 0,  ||residual|| = 0.83
Iteration 0,  ||residual|| = 0.82
Iteration 0,  ||residual|| = 0.99
Iteration 0,  ||residual|| = 0.99
Iteration 0,  ||residual|| = 0.99
Iteration 0,  ||residual|| = 0.99
Iteration 0,  ||residual|| = 1.58
Iteration 0,  ||residual|| = 1.58
Iteration 0,  ||residual|| = 1.57
Iteration 0,  ||residual|| = 1.58
2 / 3
Iteration 0,  ||residual|| = 0.56
Iteration 0,  ||residual|| = 0.56
Iteration 0,  ||residual|| = 0.56
Iteration 0,  ||residual|| = 0.56
Iteration 0,  ||residual|| = 1.02
Iteration 0,  ||residual|| = 1.02
Iteration 0,  ||residual|| = 1.02
Iteration 0,  ||residual|| = 1.02
Iteration 0,  ||residual|| = 1.16
Iteration 0,  ||residual|| = 1.16
Iteration 0,  ||residual|| = 1.16
Iteration 0,  ||residual|| = 1.16
3 / 3
Grid of 200 X 200
Iteration 0,  ||residual|| = 1.10
Iteration 0,  ||residual|| = 1.10
Iteration 0,  ||residual|| = 1.10
Iteration 0,  ||residual|| = 1.10
```

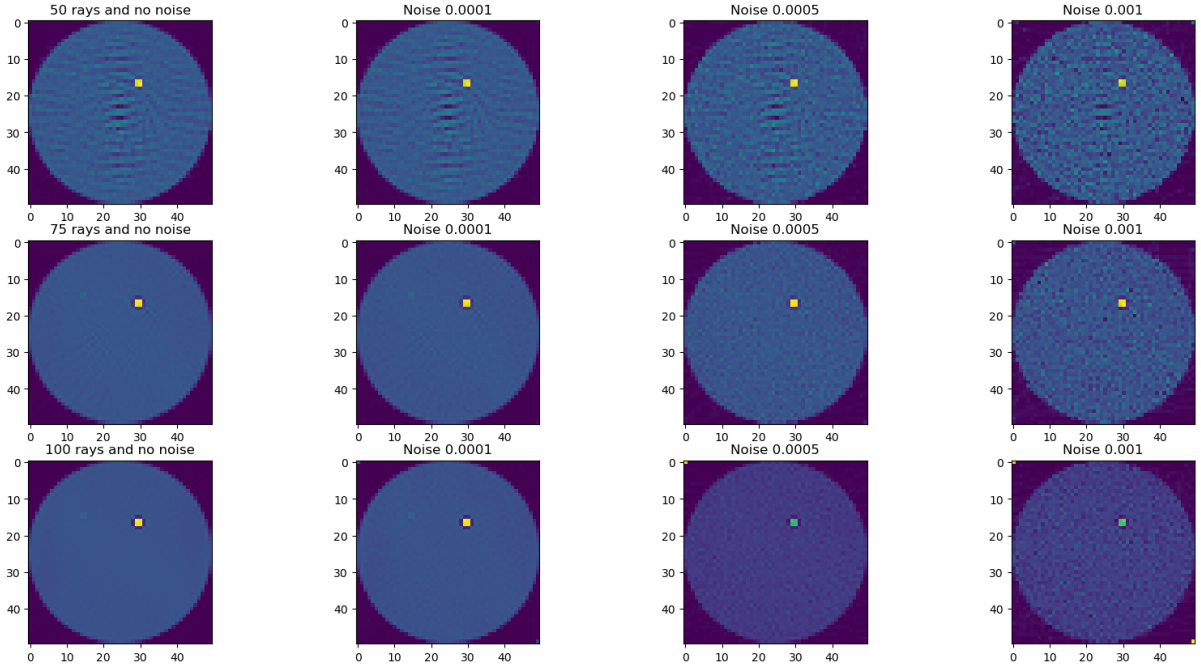
```
Iteration 0,  ||residual|| = 1.32
Iteration 0,  ||residual|| = 1.32
Iteration 0,  ||residual|| = 1.32
Iteration 0,  ||residual|| = 1.32
Iteration 0,  ||residual|| = 2.06
Iteration 0,  ||residual|| = 2.06
Iteration 0,  ||residual|| = 2.06
Iteration 0,  ||residual|| = 2.07
1 / 3
Iteration 0,  ||residual|| = 1.04
Iteration 0,  ||residual|| = 1.04
Iteration 0,  ||residual|| = 1.04
Iteration 0,  ||residual|| = 1.04
Iteration 0,  ||residual|| = 1.17
Iteration 0,  ||residual|| = 1.17
Iteration 0,  ||residual|| = 1.18
Iteration 0,  ||residual|| = 1.17
Iteration 0,  ||residual|| = 2.02
Iteration 0,  ||residual|| = 2.02
Iteration 0,  ||residual|| = 2.02
Iteration 0,  ||residual|| = 2.02
2 / 3
Iteration 0,  ||residual|| = 0.80
Iteration 0,  ||residual|| = 0.80
Iteration 0,  ||residual|| = 0.80
Iteration 0,  ||residual|| = 0.80
Iteration 0,  ||residual|| = 1.21
Iteration 0,  ||residual|| = 1.21
Iteration 0,  ||residual|| = 1.21
Iteration 0,  ||residual|| = 1.21
Iteration 0,  ||residual|| = 1.46
Iteration 0,  ||residual|| = 1.46
Iteration 0,  ||residual|| = 1.46
Iteration 0,  ||residual|| = 1.46
3 / 3
```



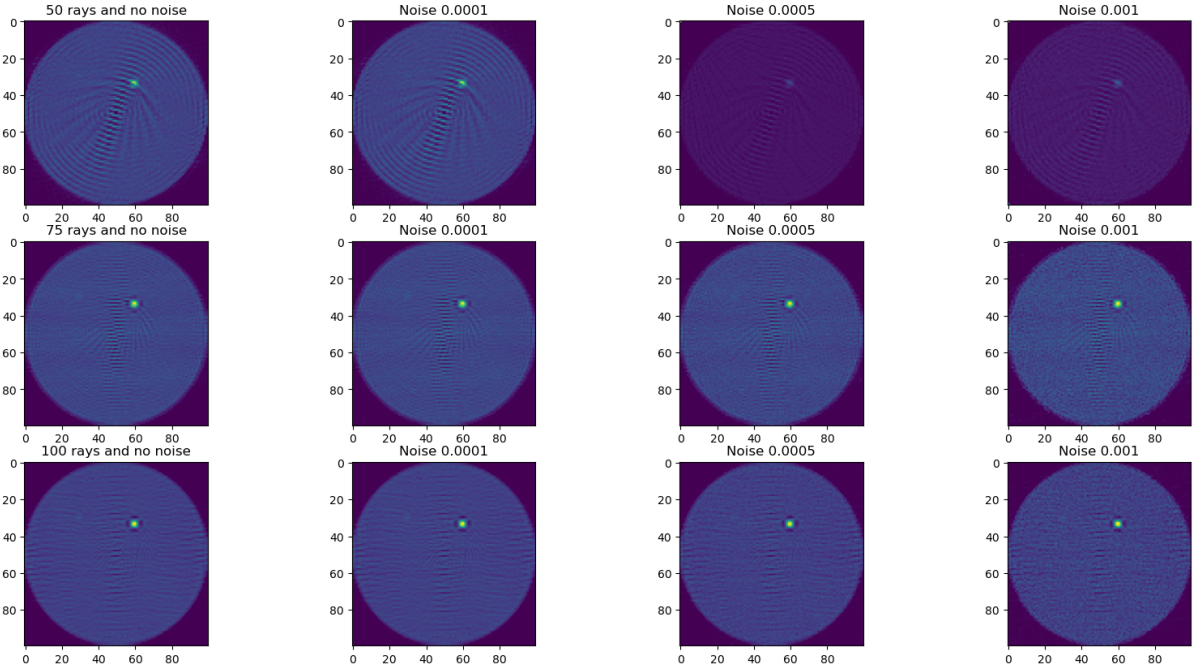
Reconstructed x (50X50) with 90 angles



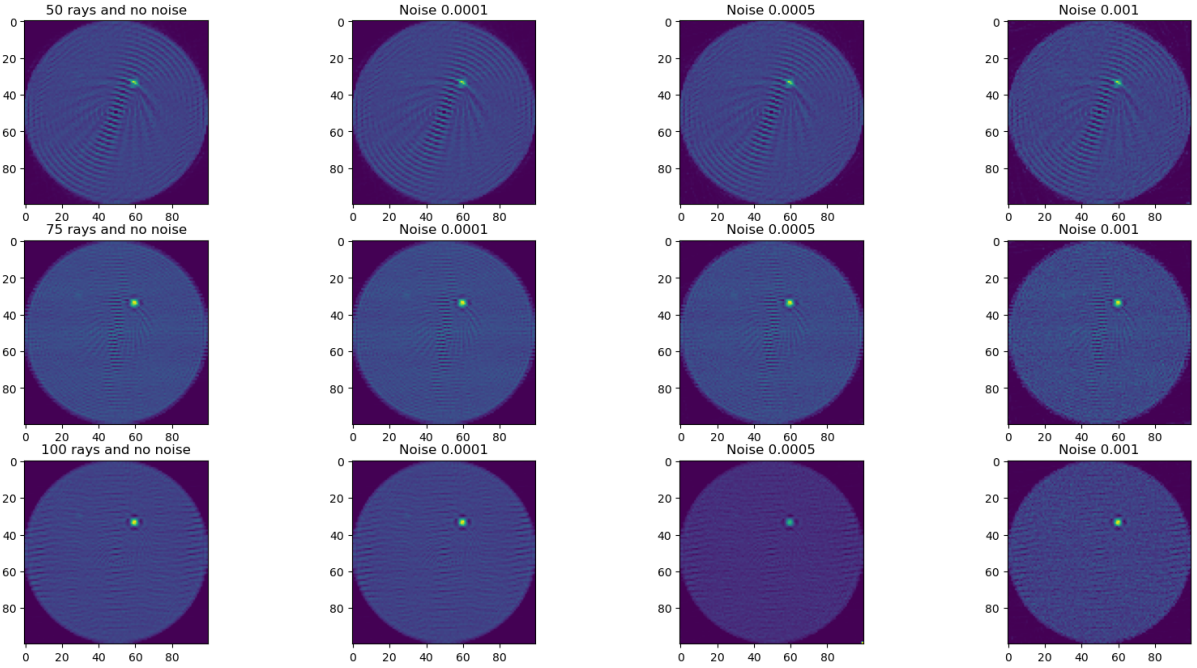
Reconstructed x (50X50) with 60 angles



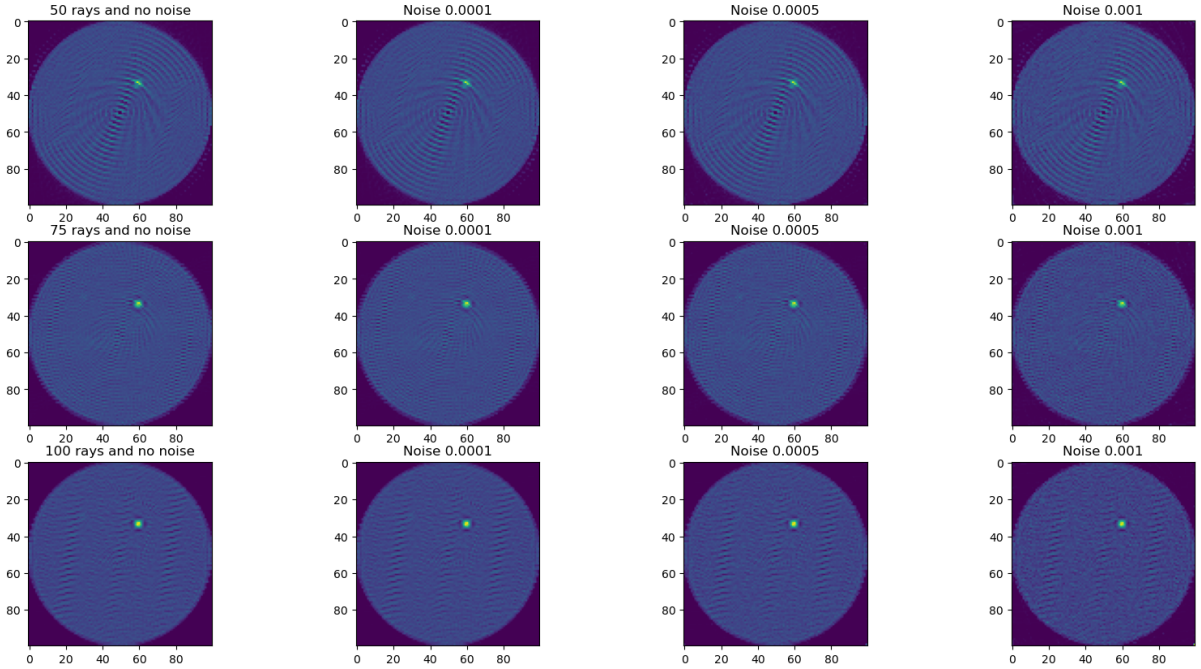
Reconstructed x (100X100) with 100 angles



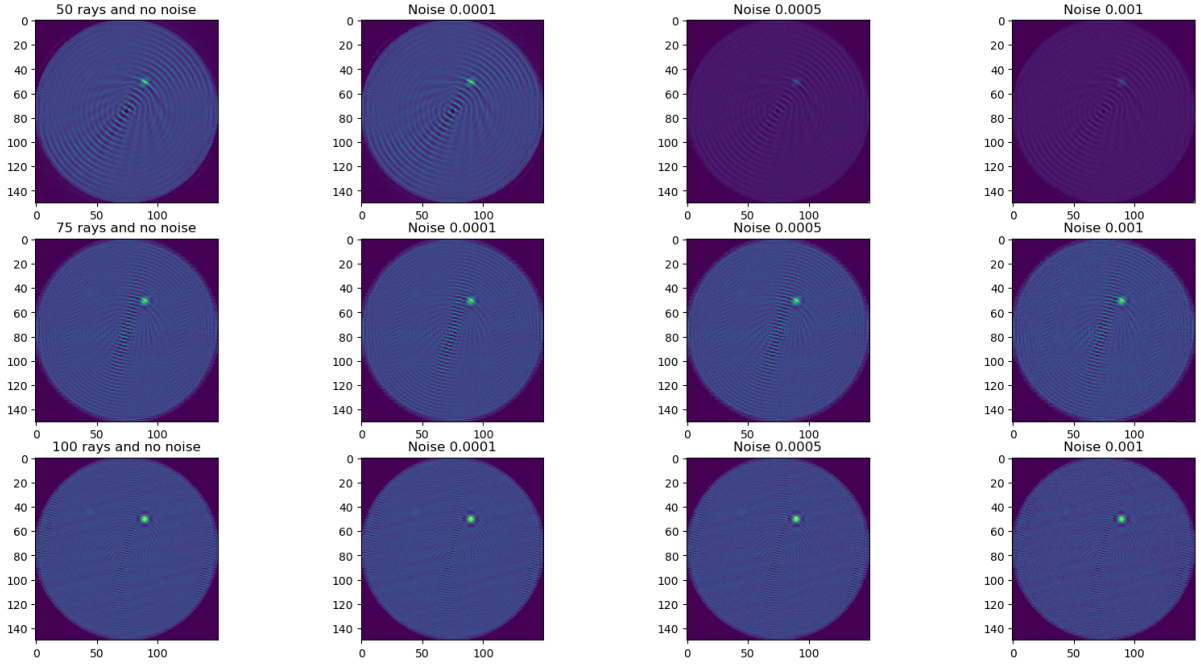
Reconstructed x (100X100) with 90 angles



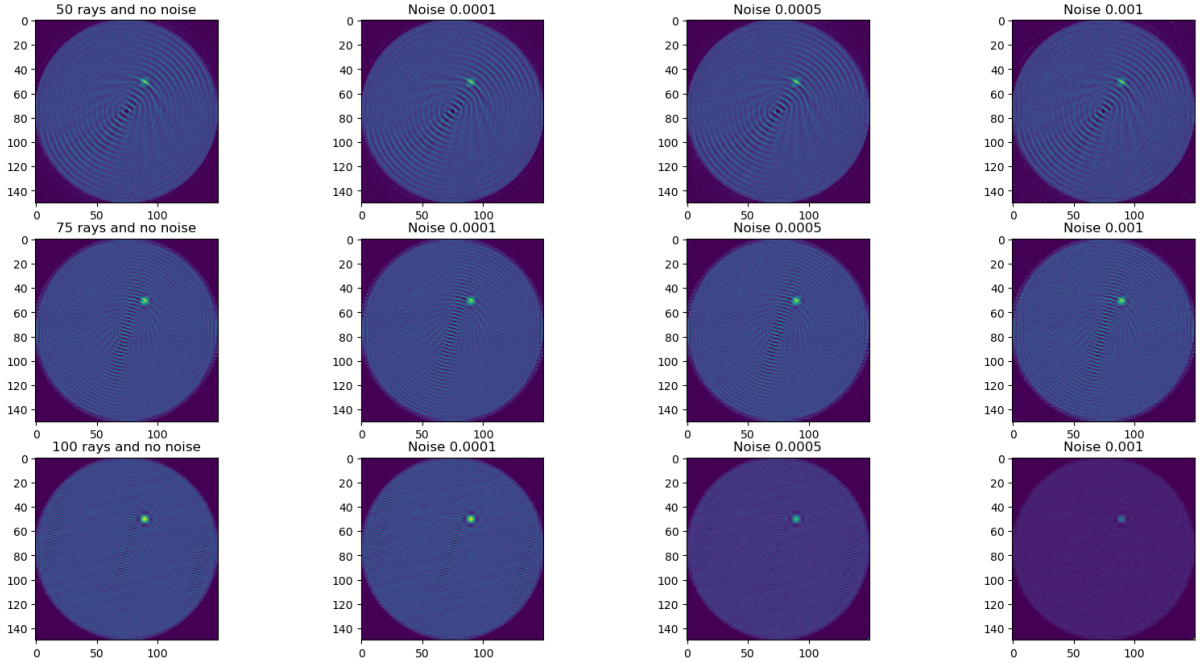
Reconstructed x (100X100) with 60 angles



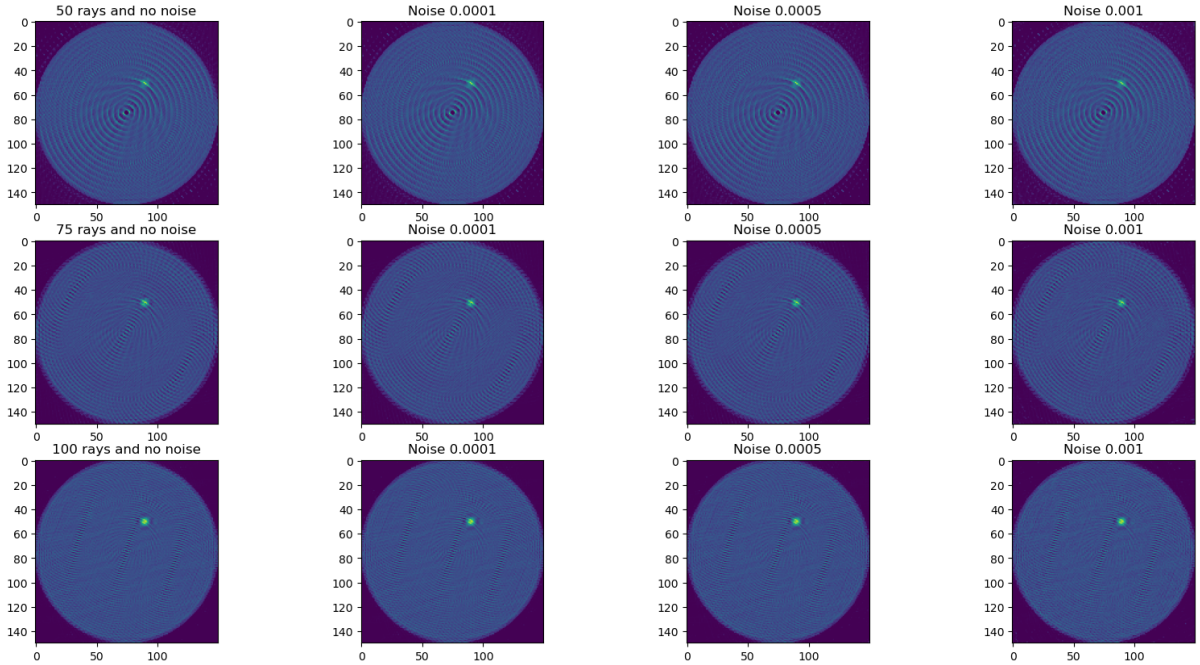
Reconstructed x (150X150) with 100 angles



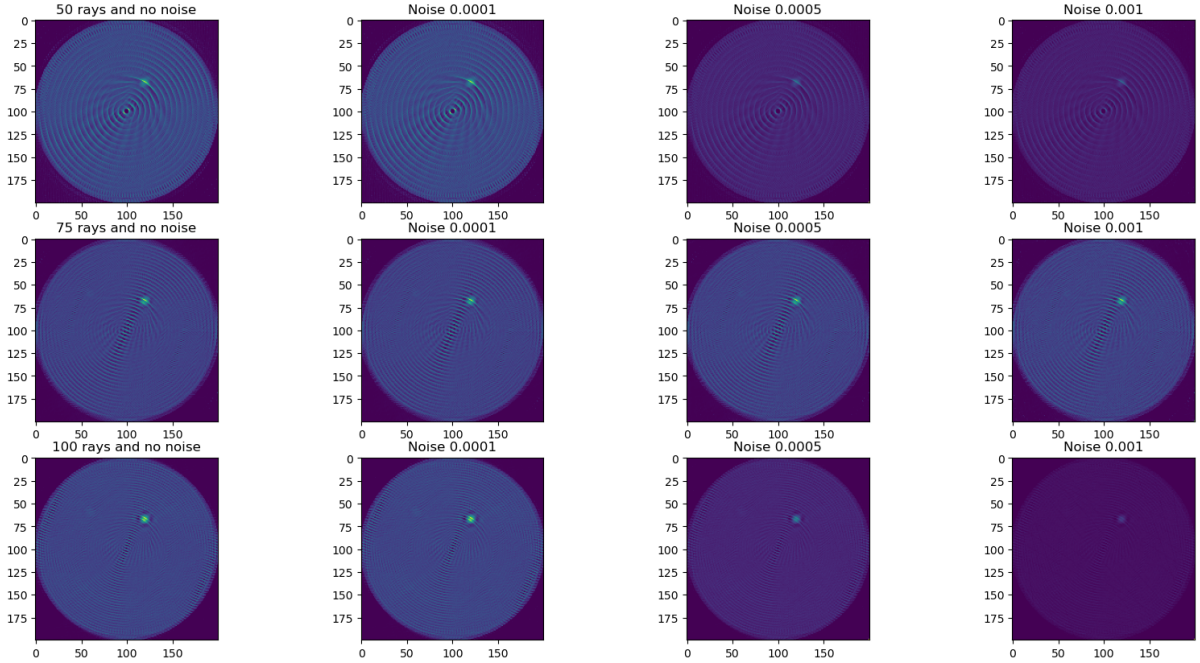
Reconstructed x (150X150) with 90 angles



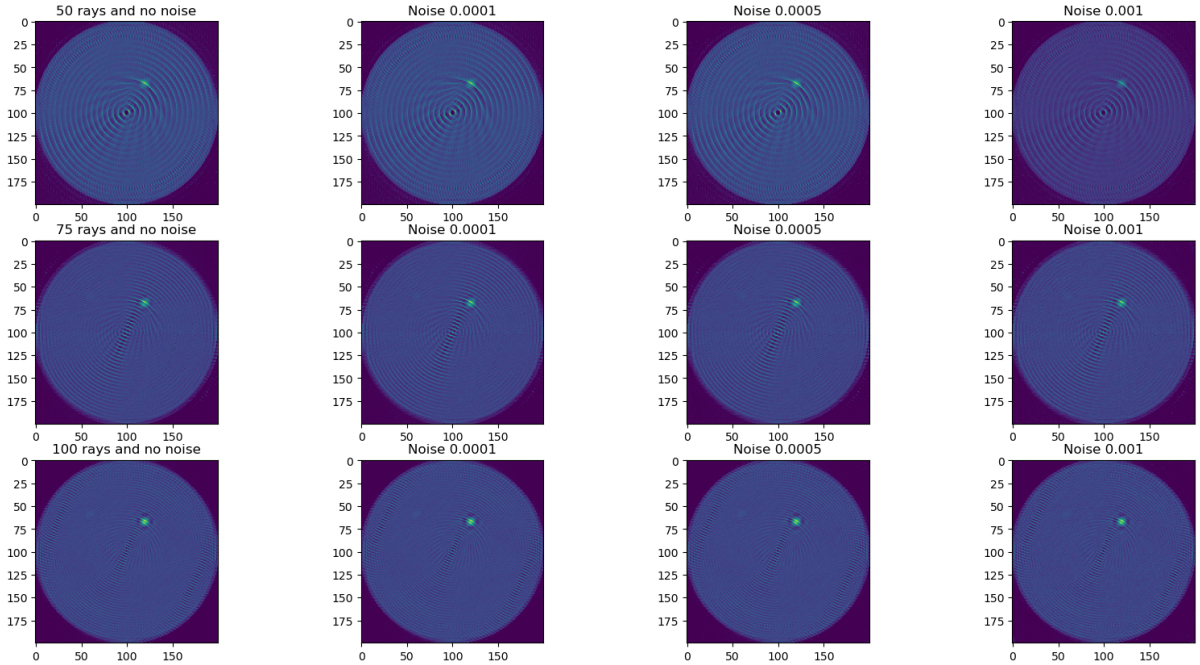
Reconstructed x (150X150) with 60 angles

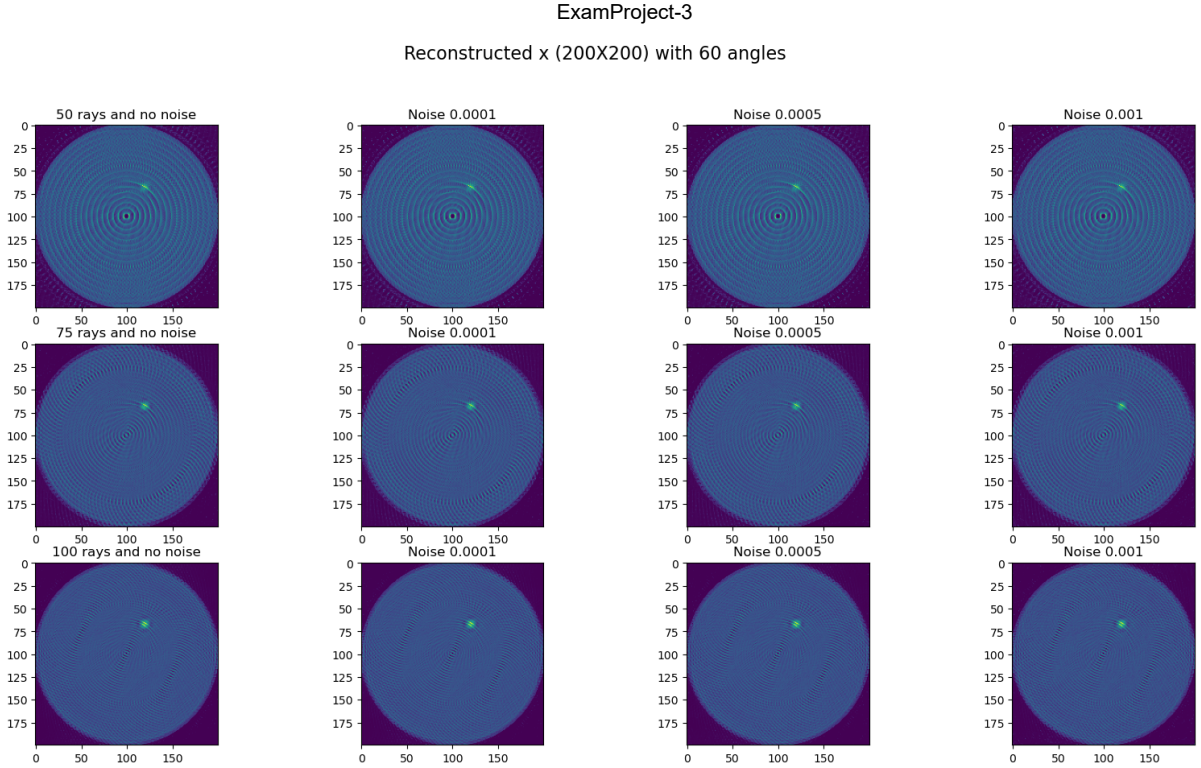


Reconstructed x (200X200) with 100 angles



Reconstructed x (200X200) with 90 angles





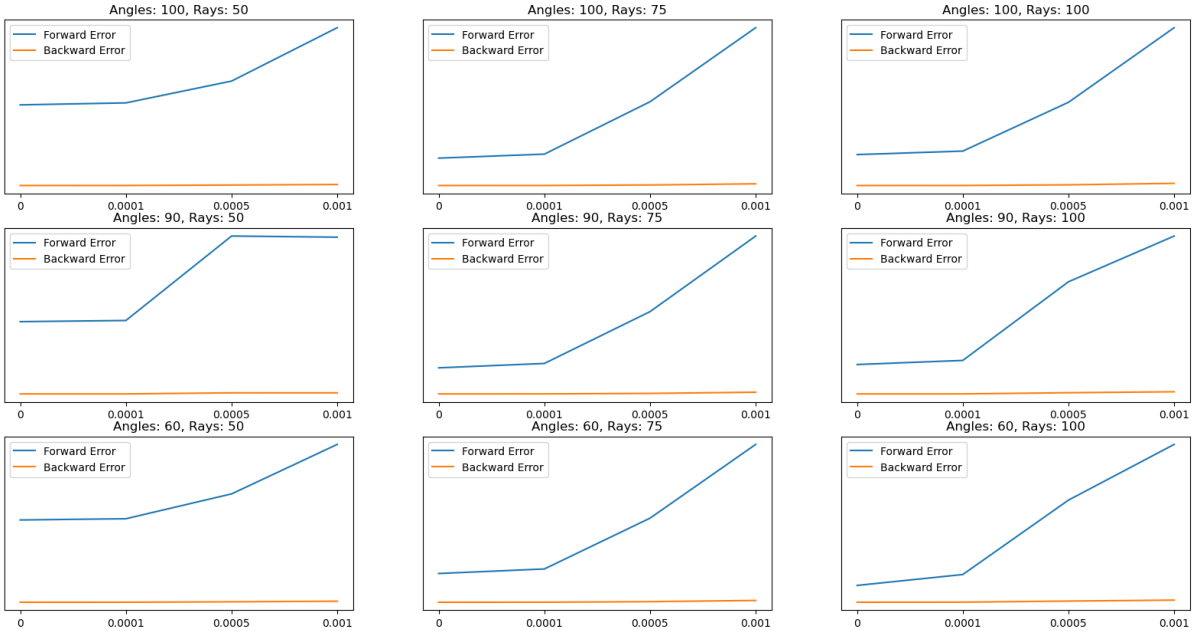
In [11]:
```
len(testA[0][0])

import numpy as np
testAS = []
for i in testA:
    for j in i:
        testj = np.array_split(j, len(j)//4)
    testAS.append(testj)
```

Plotting the errors:

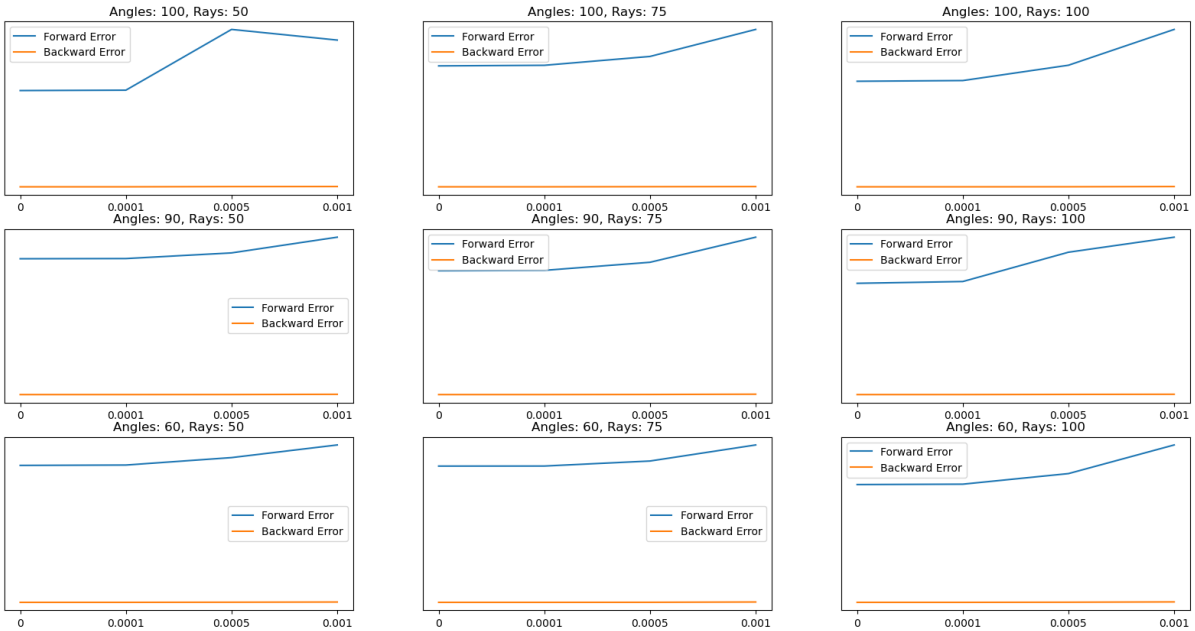
In [86]:
```
n=0
resample_x = [100,200,300,400]

angles = [100,100,100,90,90,90,60,60,60]
#rays = [50,50,50,75,75,75,100,100,100]
rays = [50,75,100,50,75,100,50,75,100]
#amount of noise
noises = [0, 1e-4, 5e-4, 1e-3]

for i in testAS:

    fig, axs = plt.subplots(3,3, figsize=(20, 10), facecolor='w', edgecolor='k')
    fig.subplots_adjust(hspace = .2)
    fig.suptitle(f'Set of {resample_x[n]/2}X{resample_x[n]/2}', fontsize=16)
    plt.setp(axs, xticks=range(len(rel_back)), xticklabels=noises,
        yticks=[1, 2, 3])
    axs = axs.ravel()
    ax = 0
    #w = 0
    for j in i:
        rel_back = [j[0][1],j[1][1], j[2][1], j[3][1]]
        rel_for = [j[0][2],j[1][2], j[2][2], j[3][2]]
        x = range(len(rel_back))
        axs[ax].plot(x,rel_for, label='Forward Error')
```

```
        axs[ax].plot(x,rel_back, label='Backward Error')
        #xticks(noises)
        axs[ax].legend()
        axs[ax].set_title(f'Angles: {angles[ax]}, Rays: {rays[ax]}')


        ax+=1

    n+=1
```

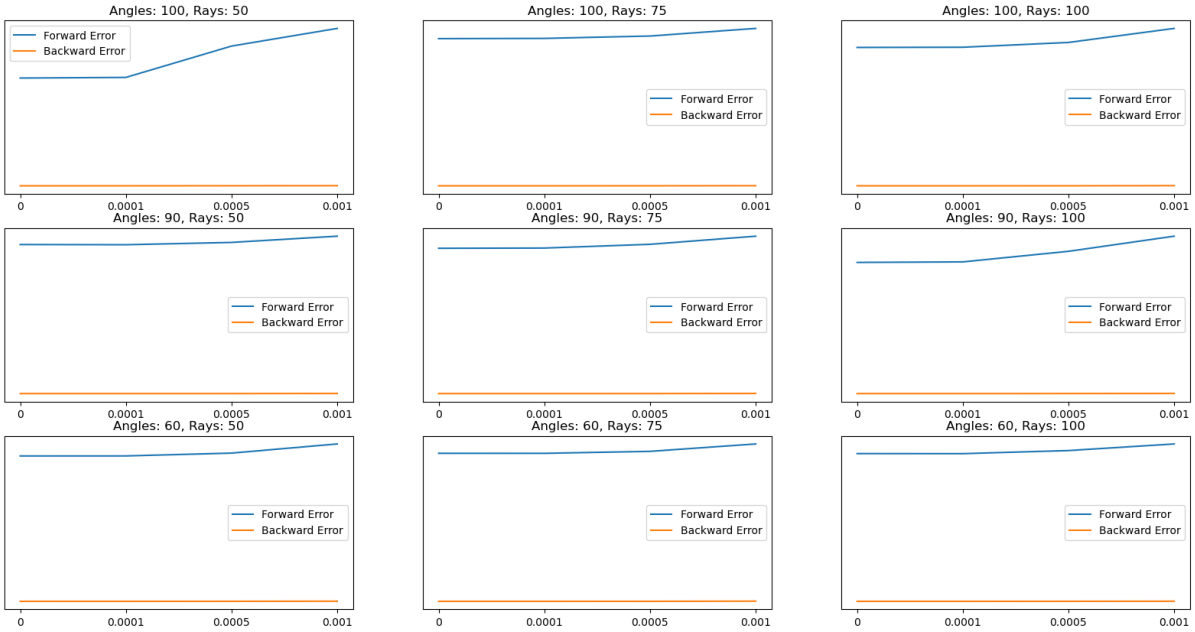
Set of 50.0X50.0



Set of 100.0X100.0



Set of 150.0X150.0



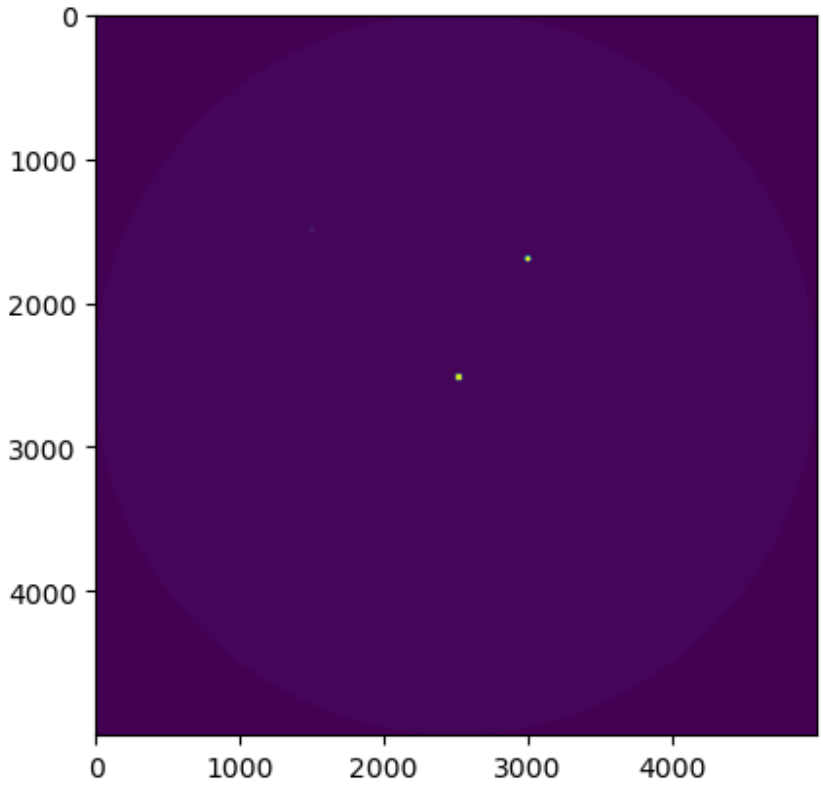
Set of 200.0X200.0

Testing on Artificial Data

In [64]:
```
import numpy as np

# Load the NumPy array from the file
array_data = np.load('testImage.npy')

# Define the material value to extract
material_value = 0.05397851616202787

# Find the indices of the material value
indices = np.where(array_data == material_value)
```

```
# Extract the shape of the material
min_x, min_y = np.min(indices, axis=1)
max_x, max_y = np.max(indices, axis=1)
material_shape = (max_x - min_x + 1, max_y - min_y + 1)

# Choose a new position to add the extracted shape
new_position = (2500,2500)

# Assign the material value to the new position in the array
array_data[new_position[0]:new_position[0]+material_shape[0],
           new_position[1]:new_position[1]+material_shape[1]] = material_value

# Get the unique values and their counts
unique_values, value_counts = np.unique(array_data, return_counts=True)

# Print the unique values and their counts
for value, count in zip(unique_values, value_counts):
    print(f"Value: {value}, Count: {count}")

# Print the modified array
#img = Image.fromarray(array_data)
#img = img.resize((50,50), Image.LANCZOS)
plt.imshow(array_data)
plt.show()
```

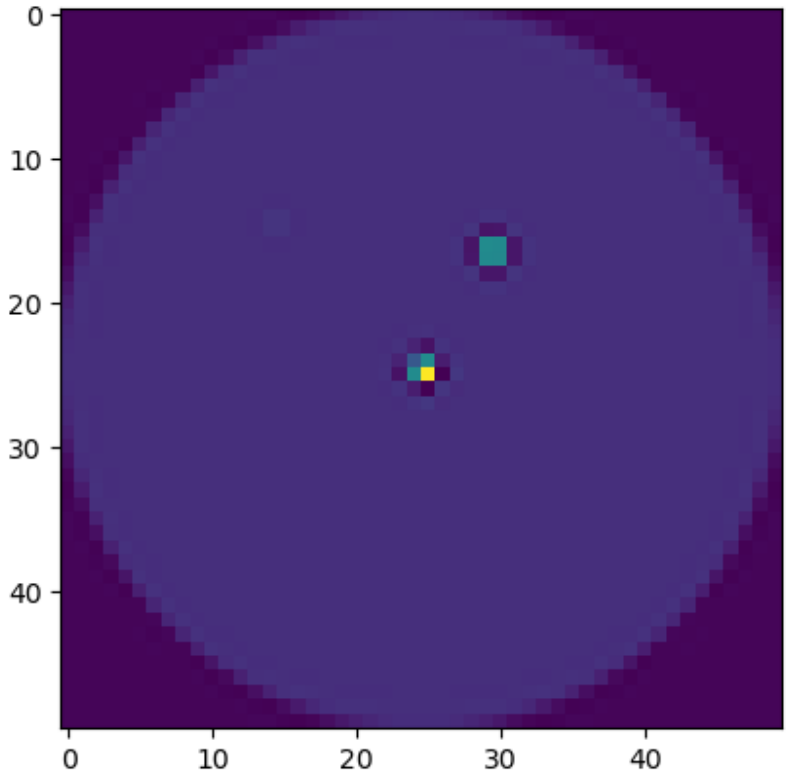
```
Value: 0.0, Count: 5365197
Value: 0.0008496388523324683, Count: 19630608
Value: 0.0037101005066195736, Count: 1257
Value: 0.05397851616202787, Count: 2938
```



In [65]:
```
imgA = Image.fromarray(array_data)
# Resize the image from 5000x5000 pixels to 50x50 pixels
imgA = imgA.resize((50,50), Image.LANCZOS)
```

```
# Convert image again from PIL to numpy
img_array = np.array(imgA)
plt.imshow(img_array)
plt.show()
```

```
C:\Users\Owner\AppData\Local\Temp\ipykernel_13468\1641601906.py:3: DeprecationWarnin
g: LANCZOS is deprecated and will be removed in Pillow 10 (2023-07-01). Use Resamplin
g.LANCZOS instead.
  imgA = imgA.resize((50,50), Image.LANCZOS)
```



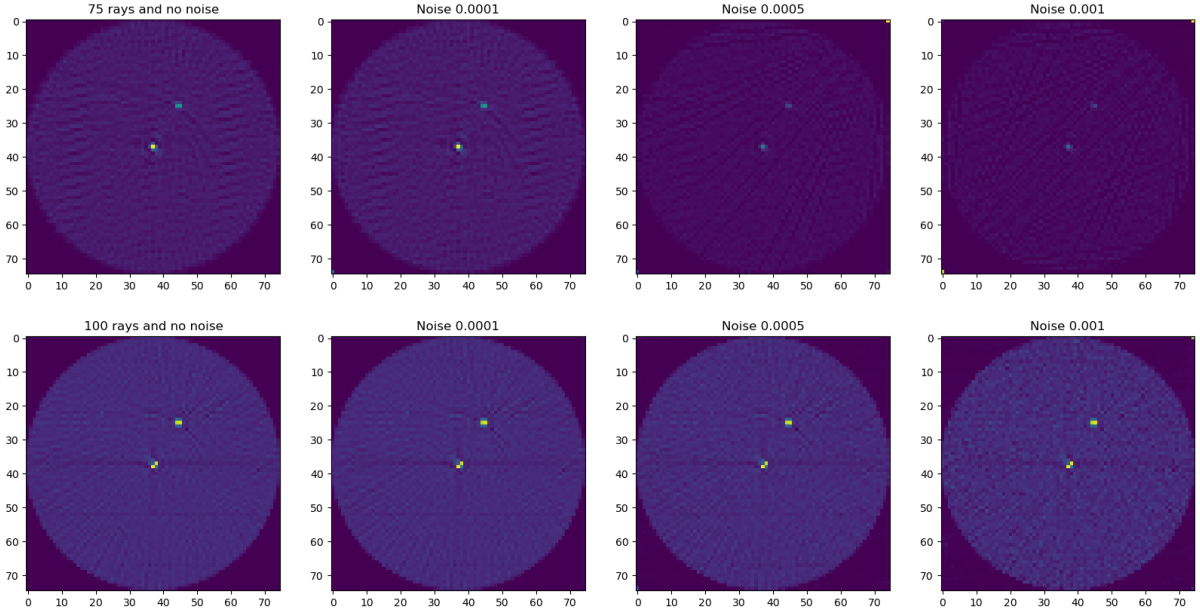
In [88]:
```
# Load the image with lead and steel shot
import matplotlib.pyplot as plt
import airtools

# Load image and convert it to a PIL Image object
img_array = array_data
img = Image.fromarray(img_array)

#Parameters to resample x, the values divided by 2 give N, to form an NXN grid
# for example, first value 50 will give a 25x25 grid
#modify and/or add as you want, just note that the bigger the grid the more it will lo

resample_x = [150,200,250]

testA1 = []

for i in resample_x:

    L = int(i/2)
    # Resize the image from 5000x5000 pixels to 50x50 pixels
    img = img.resize((L, L), Image.Resampling.LANCZOS)

    # Convert image again from PIL to numpy
    img_array = np.array(img)
```

```
x=img_array.flatten()
N = img_array.shape[0]
print(f'Grid of {L} X {L}')

#You can modify and/or add more variations for the next parameters


#These are the number of angles that we may use, we will later on use np.linspace(
# values between 0 and 180

angles = [50,60]

#number of rays for each angle
rays = [75,100]

#amount of noise
noises = [1e-4, 5e-4, 1e-3]

#ignore
pt=0
testA11 = []

for j in angles:

    #create figure for subplots for each number of angles
    fig, axs = plt.subplots(len(rays),len(noises)+1, figsize=(20, 10), facecolor=
    fig.subplots_adjust(hspace = .2)
    fig.suptitle(f'Reconstructed x ({L}X{L}) with {j} angles', fontsize=16)
    axs = axs.ravel()
    ax = 0

    for n in rays:

        #set parameters for paralleltomo

        theta_i = np.matrix(np.linspace(0,180,j))
        p=n

        #paralleltomo
        [A,thetar,pr,d] = paralleltomo.paralleltomo(N, theta_i, p)

        #obtain b
        b=np.dot(A,x)

        A, b, som = airtools.rzr(A,b)
        #condA = np.linalg.cond(A)
        x_reconstructed, residuals = airtools.kaczmarz(A,b)
        condA = np.linalg.cond(A, 2)
        #print(f'condition number: {cond}')

        r_v = (A@x_reconstructed)-b
        A_inv = np.linalg.pinv(A)
        deltax_norm= (np.linalg.norm(-A_inv@r_v))
        rel_res = (np.linalg.norm(r_v))/((np.linalg.norm(A))*(np.linalg.norm(x)))
        rel_error = (np.linalg.norm(deltax_norm))/(np.linalg.norm(x))
        #EMF = rel_error/rel_res
        #print(f'Condition number of A with N = {N}, nA = {j}, p = {p}: {condA}')
        #print(f'Relative error of reconstructed x: {rel_error}')
        #print(f'Relative residual: {rel_res} ')
        testA11.append([condA, rel_res, rel_error])
```

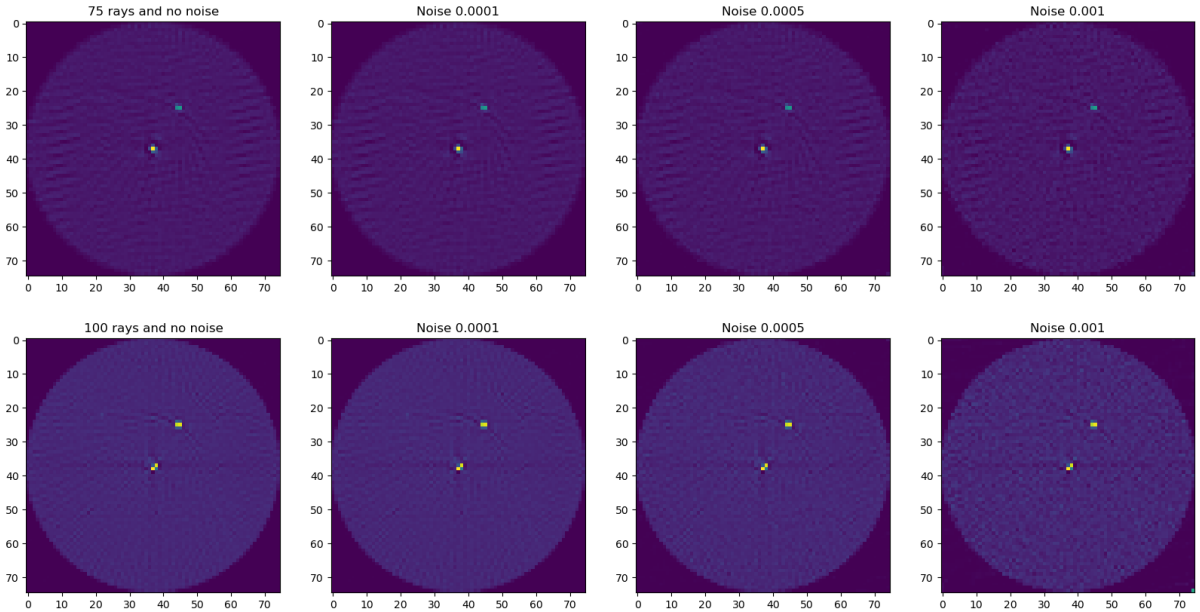
```
            #x_reconstructed, residuals = airtools.kaczmarz(A,b)

            orig_shape = (N, N)
            # reshape the flattened image to the original shape
            recovered_img_x = x_reconstructed.reshape(orig_shape)
            # convert the image back to PIL Image format
            recovered_img_x = np.array(recovered_img_x)


            axs[ax].imshow(recovered_img_x )
            axs[ax].set_title(f'{p} rays and no noise')
            #sensitivity = np.linalg.cond(A)


            ax+=1
            for a in noises:

              #ADD NOISE
                noise=np.random.normal(0,a, b.shape)
                b_noise= b + noise
                A, b_noise, som = airtools.rzr(A,b_noise)

                x_noise, residuals = airtools.kaczmarz(A,b_noise)

                r_v = (A@x_noise)-b
                A_inv = np.linalg.pinv(A)
                deltax_norm= (np.linalg.norm(-A_inv@r_v))
                rel_res = (np.linalg.norm(r_v))/((np.linalg.norm(A))*(np.linalg.norm(x
                rel_error = (np.linalg.norm(deltax_norm))/(np.linalg.norm(x))
                EMF = rel_error/rel_res
                #print(f'EMF with noise = {a}: {EMF}')
                #print(f'Relative error of reconstructed x: {rel_error}')
                #print(f'Relative residual: {rel_res} ')
                testA11.append([EMF, rel_res, rel_error])

                # reshape the flattened image to the original shape
                recovered_img_x_noise = x_noise.reshape(orig_shape)
                # convert the image back to PIL Image format
                recovered_img_x_noise = np.array(recovered_img_x_noise)

                axs[ax].imshow(recovered_img_x_noise)
                axs[ax].set_title(f'Noise {a}')
                ax+=1
        pt+=1
        print(f'{pt} / {len(angles)}')
        #pt+=1

    testA1.append([testA11])
```

```
Grid of 75 X 75
Iteration 0,  ||residual|| = 0.42
Iteration 0,  ||residual|| = 0.42
Iteration 0,  ||residual|| = 0.44
Iteration 0,  ||residual|| = 0.45
Iteration 0,  ||residual|| = 0.21
Iteration 0,  ||residual|| = 0.21
Iteration 0,  ||residual|| = 0.21
Iteration 0,  ||residual|| = 0.21
1 / 2
Iteration 0,  ||residual|| = 0.47
Iteration 0,  ||residual|| = 0.47
Iteration 0,  ||residual|| = 0.46
Iteration 0,  ||residual|| = 0.48
Iteration 0,  ||residual|| = 0.23
Iteration 0,  ||residual|| = 0.23
Iteration 0,  ||residual|| = 0.23
Iteration 0,  ||residual|| = 0.24
2 / 2
Grid of 100 X 100
Iteration 0,  ||residual|| = 0.66
Iteration 0,  ||residual|| = 0.66
Iteration 0,  ||residual|| = 0.67
Iteration 0,  ||residual|| = 0.66
Iteration 0,  ||residual|| = 0.65
Iteration 0,  ||residual|| = 0.65
Iteration 0,  ||residual|| = 0.65
Iteration 0,  ||residual|| = 0.65
1 / 2
Iteration 0,  ||residual|| = 0.74
Iteration 0,  ||residual|| = 0.74
Iteration 0,  ||residual|| = 0.74
Iteration 0,  ||residual|| = 0.74
Iteration 0,  ||residual|| = 0.69
Iteration 0,  ||residual|| = 0.69
Iteration 0,  ||residual|| = 0.69
Iteration 0,  ||residual|| = 0.69
2 / 2
Grid of 125 X 125
Iteration 0,  ||residual|| = 0.80
Iteration 0,  ||residual|| = 0.80
Iteration 0,  ||residual|| = 0.80
Iteration 0,  ||residual|| = 0.80
Iteration 0,  ||residual|| = 0.96
Iteration 0,  ||residual|| = 0.96
Iteration 0,  ||residual|| = 0.96
Iteration 0,  ||residual|| = 0.97
1 / 2
Iteration 0,  ||residual|| = 0.92
Iteration 0,  ||residual|| = 0.92
Iteration 0,  ||residual|| = 0.92
Iteration 0,  ||residual|| = 0.92
Iteration 0,  ||residual|| = 1.01
Iteration 0,  ||residual|| = 1.01
Iteration 0,  ||residual|| = 1.01
Iteration 0,  ||residual|| = 1.01
2 / 2
```

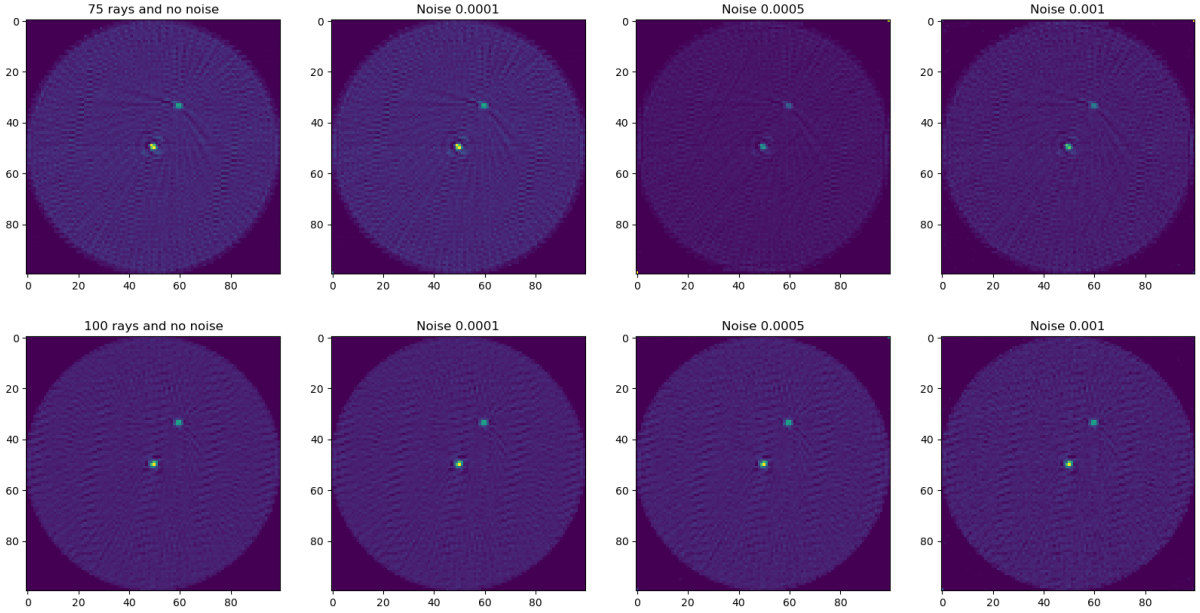
Reconstructed x (75X75) with 50 angles



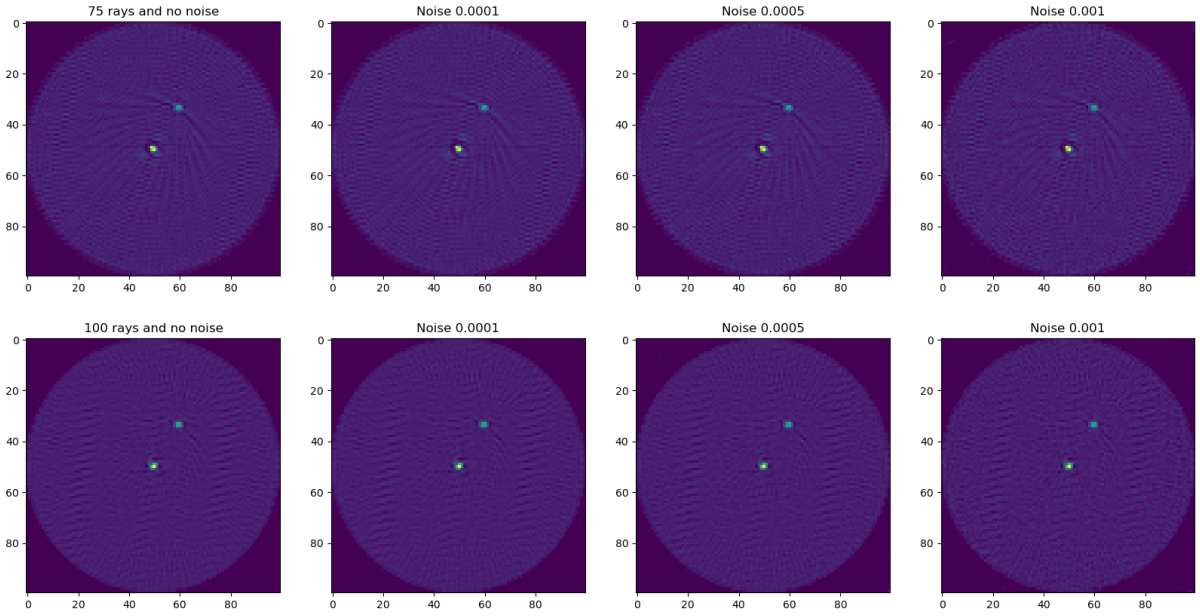
Reconstructed x (75X75) with 60 angles

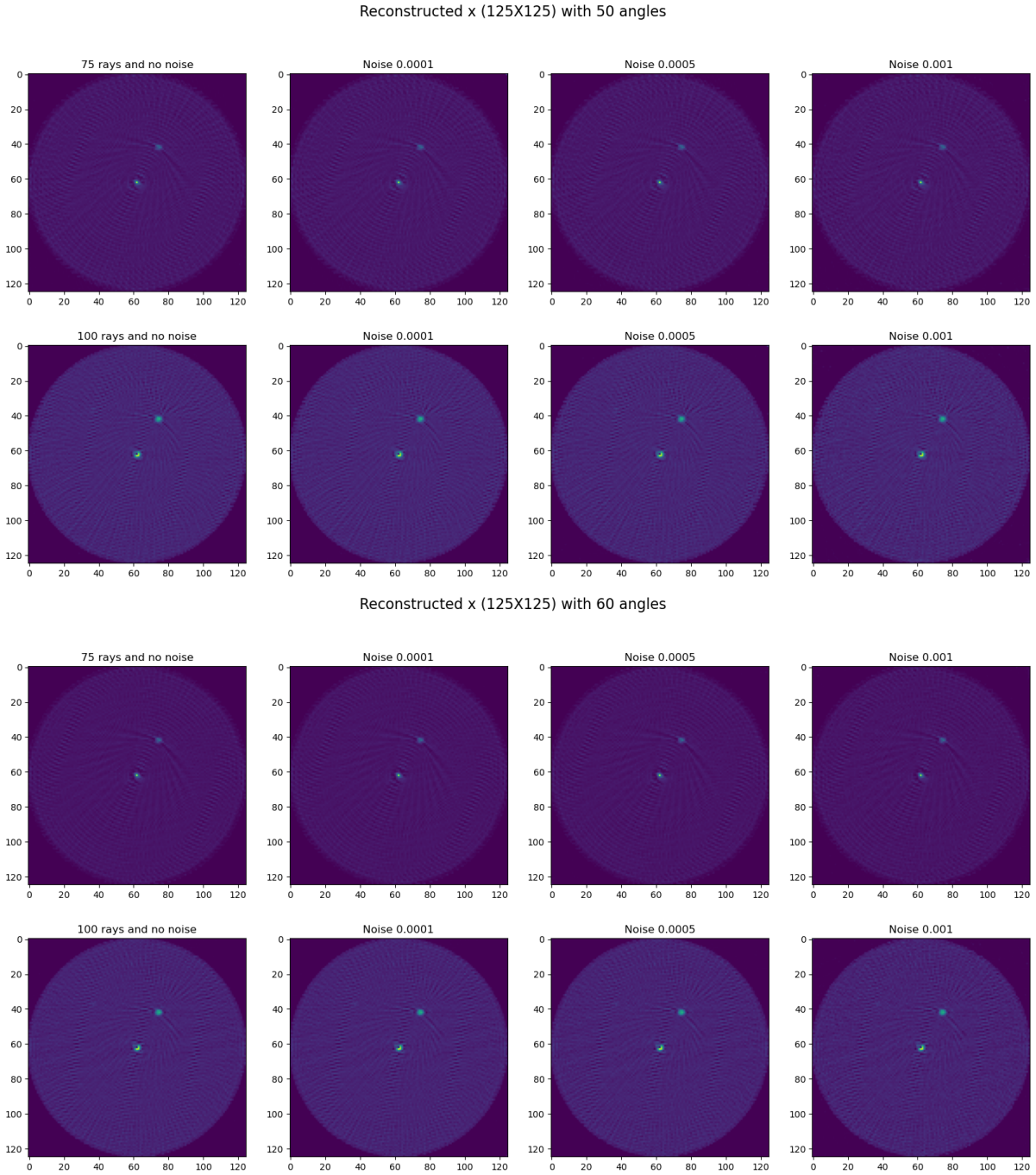


Reconstructed x (100X100) with 50 angles



Reconstructed x (100X100) with 60 angles



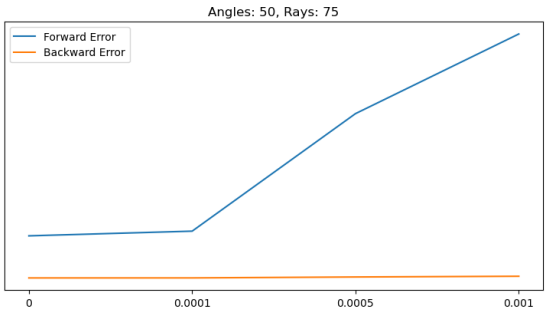


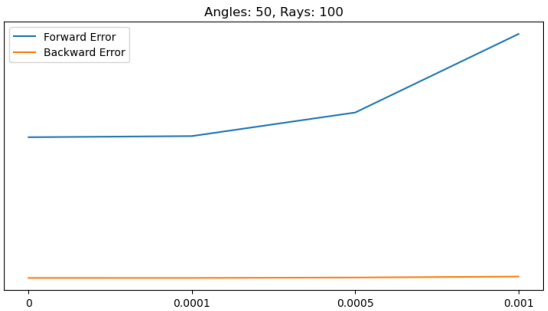
Plotting the errors:

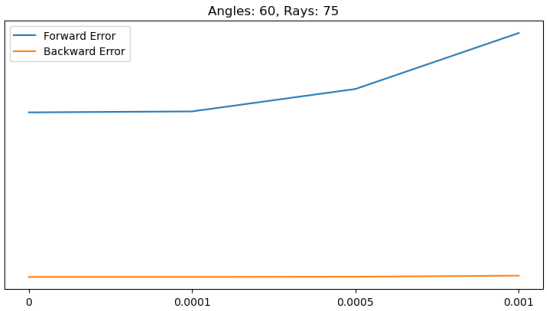
In [95]:
```
len(testA1[0][0])

import numpy as np
testAS1 = []
for i in testA1:
    for j in i:
        testj = np.array_split(j, len(j)//4)
    testAS1.append(testj)

n=0

resample_x = [150,200,250]
angles = [50,50,60,60]
```

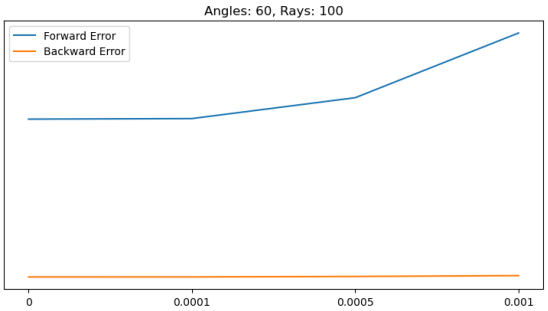
```
rays = [75,100,75,100]

noises = [0,1e-4, 5e-4, 1e-3]

#resample_x = [100,200,300,400]

#angles = [100,100,100,90,90,90,60,60,60]
#rays = [50,50,50,75,75,75,100,100,100]
#rays = [50,75,100,50,75,100,50,75,100]
#amount of noise
#noises = [0, 1e-4, 5e-4, 1e-3]

for i in testAS1:

    fig, axs = plt.subplots(2,2, figsize=(20, 10), facecolor='w', edgecolor='k')
    fig.subplots_adjust(hspace = .2)
    fig.suptitle(f'Set of {resample_x[n]/2}X{resample_x[n]/2}', fontsize=16)
    plt.setp(axs, xticks=range(len(rel_back)), xticklabels=noises,
        yticks=[1, 2, 3])
    axs = axs.ravel()
    ax = 0
    #w = 0
    for j in i:
        rel_back = [j[0][1],j[1][1], j[2][1], j[3][1]]
        rel_for = [j[0][2],j[1][2], j[2][2], j[3][2]]
        x = range(len(rel_back))
        axs[ax].plot(x,rel_for, label='Forward Error')
        axs[ax].plot(x,rel_back, label='Backward Error')
        #xticks(noises)
        axs[ax].legend()
        axs[ax].set_title(f'Angles: {angles[ax]}, Rays: {rays[ax]}')


        ax+=1

    n+=1
```

Set of 75.0X75.0

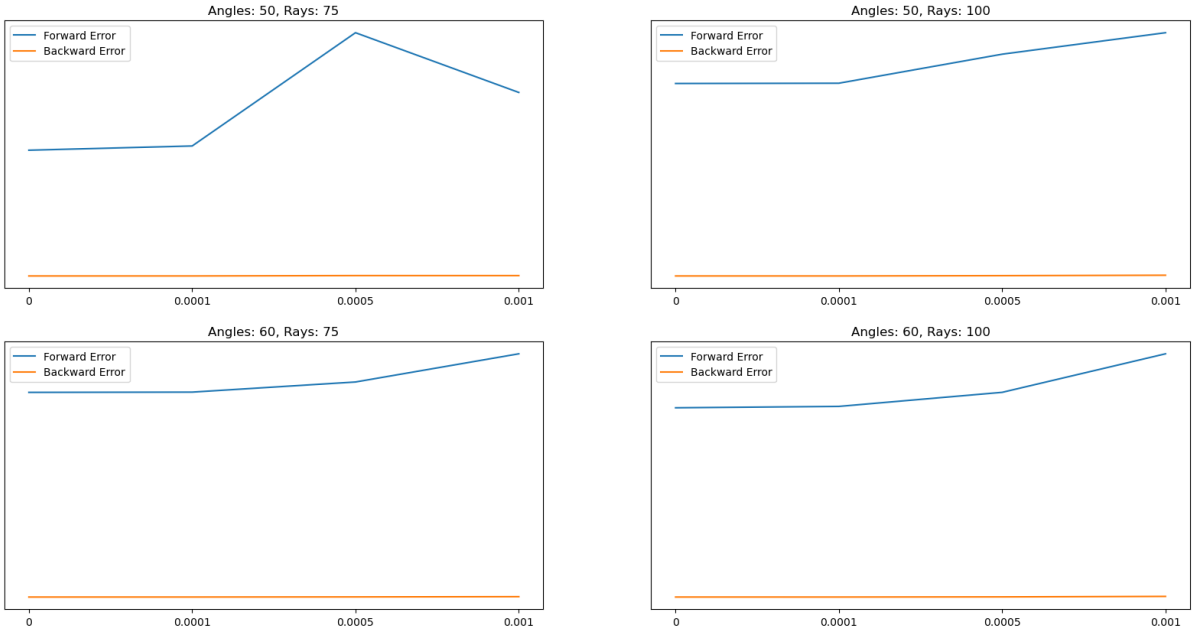




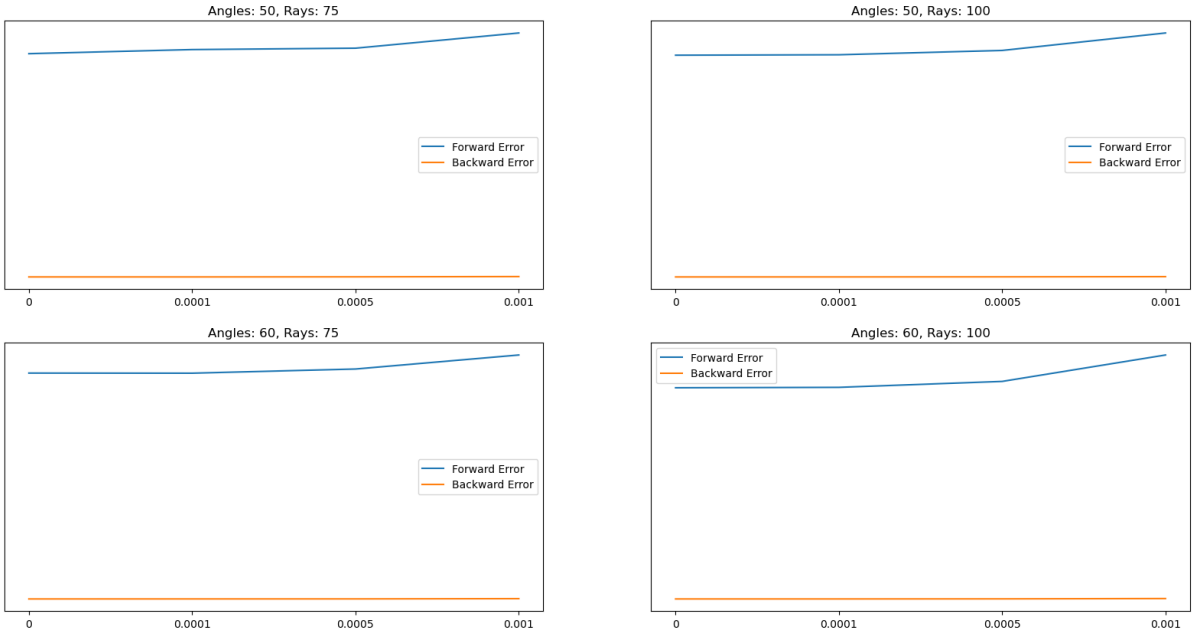




Set of 100.0X100.0



Set of 125.0X125.0



http://www2.compute.dtu.dk/~pcha/AIRtoolsII/

https://reader.elsevier.com/reader/sd/pii/S0377042711005188?token=88D5903D423249F3954638A56A5CFCE9599BDC95CC84154B3BDBCACAE3FCEFA8CEBB98B8&west-1&originCreation=20230411085058

https://backend.orbit.dtu.dk/ws/portalfiles/portal/140598691/10.1007_2Fs11075_017_0430_x.pdf?fbclid=IwAR0kygJvFsCW7OFLcLnuL3T7FHpReAlFuU9XE8iGP96GRH2AXMbKybgyTMs

http://people.compute.dtu.dk/pcha/AIRtoolsII/