Universiteit Leiden

Master Thesis Mathematics

Optimisation of projection angle selection in Computed Tomography

Author:
Francien G. Bossema, BSc.

Supervisors:
Dr. Floske Spieksma (UL)
Prof. dr. Joost Batenburg (UL/CWI)
Dr. Felix Lucka (CWI)
Dr. Alexander Kostenko (CWI)

August 30, 2018

Contents

Abstract

In Computed Tomography, it is often standard to scan an object with the projection angles spread equidistantly over the full rotation of the object. If the number of projection angles is relatively small (e.g. because of radiation damage), the choice of the distribution of these angles is important. Especially if the object has certain main directions, in which the most important features are aligned, the reconstruction can be improved by selecting angles in these directions. In this thesis, possibilities for improving the choice of projection angles are investigated. Three angle selection algorithms are discussed: a greedy algorithm, a coordinate descent algorithm and an adaptation of the ensemble Kalman Filter algorithm. The algorithms try to find the minimum of a cost function, which is based on the L^2-norm. The performance of these algorithms is shown on three computer generated phantoms and one real-world image, generated from scanning a wooden tree stem in the FleX-Ray scanner at CWI. The results of choosing angles with the algorithms are compared with choosing equidistant projection angles. The results show that especially the coordinate descent method is capable of finding projection angles that lower the cost function. In real life situations the true image is not available. Therefore, the possibility of using training data to estimate the cost function in that case, is investigated. We assume that these training samples come from the same probability distribution as the true image. Then, averaging over the cost function of these training samples improves the choice of projection angles with respect to equidistantly chosen angles.

1 Introduction

The word *tomography* is derived from the Greek words *tomos* (slice or section) and *graphein* (to draw or to write). In Computed Tomography (CT) the goal is to reconstruct an image from several X-ray photos. X-rays can be used to look inside objects that our eyes are not able to penetrate and are therefore very useful in medical applications, where it is important to see inside a patient without having to operate. CT is therefore called a *non-invasive* method.

For many people, their first association with X-rays is often a medical application. However, CT is used for imaging in a wide range of applications. Because of the non-invasive character of CT, it is not necessary to cut an object to look into it. In the timber industry CT is used to assess the quality of wood and to detect internal defects [10]. Knots and branches become visible, but also rottenness can be detected [16]. For wood, X-ray scanning has another purpose as well: dendrochronology. In dendrochronology, the age of wood is estimated using the year rings. Because of the non-destructive character of CT, application of CT to dendrochronology will allow more objects to be dated [5].
CT has been successfully used in other research fields such as biology, archaeology and soil science. It is moreover used for industrial purposes, e.g. defect detection, and in aviation security, e.g. luggage inspection [16].
Another interesting application is the (art) historic research. Musea are interested in scanning some of their objects, for research and conservation purposes [26]. Lately, mummy cases, the decorated boxed in which mummified bodies were placed, have been scanned to reveal writing on the recycled paper they were made of [13] [27]. In 2015, the British Museum scanned an enormous mummified crocodile from ancient Egypt. Thanks to CT imaging, information on mummification procedures was obtained, as well as the crocodile's last meal [25].
Another example is the bronze statue, called 'The Youth of Magdalensberg'. Examination using CT revealed that the wall thickness of the figure was much thicker than usual in Roman techniques. Thus it became clear that the statue was not the original, but a Renaissance copy [16].

To illustrate the underlying reconstruction problem in CT, we will give an example using visible light. Imagine shining a light on an object that is located behind a screen. So, one is able to see the shadow of the object, but not the object itself. How can one determine what object is behind the screen? There can be several 3D shapes that have the same 2D shadow and thus the answer is not unique. An approach for this problem can be to rotate the object and look at its shadow from several points of view. In that way, with each different angular position of the object, more information is gained on its 3D shape, by looking at the shadow corresponding to that position.
In X-ray tomography a similar problem arises. When an X-ray image is taken, the rays are (partly) blocked by the object. Certain parts are darker than others, because the X-rays can penetrate some materials more easily than others. This gives a 2D *projection image* of a 3D object. In CT the projection image depends on the *attenuation coefficient* of the material, which depends on the density of the material and how much the intensity of the X-ray beam decreases, while travelling through the material. From one projection image, there is no unique way to determine the original shapes and attenuation coefficients of the materials inside an object. Certain features may be clear in one projection direction, but unclear in another projection direction. It is therefore useful to create multiple projections, to combine the information of all in order to estimate the original object with a high probability. Reconstruction of the original object from multiple X-ray projections is the goal of Computed Tomography.

The more projections, the clearer the internal structure of the object becomes. The object often rotates on a platform between the X-ray source and the detector. Sometimes it is not possible to take projections from all sides, for example because the size of the object does not allow for a full rotation within the scanner. In other cases it might be necessary to take as few projections as possible, due to the exposure to radiation. A common way to select the distribution of rotation angles, also called *projection angles*, is to divide the full rotation in regular angular intervals. For example by scanning each degree (360 projection images) or each $\frac{1}{3}$-th degree (1080 projection images). Especially, in the case that the number of possible projections is limited, taking equidistant projection angles can be sub-optimal. This is mainly the case, when the object has 'main orientations'. A simple example are the two main orientations of a rectangle, one along each side. For a quadrangle the angles of which are not necessarily 90 degrees, a *diamond*, choosing the projection angles along the sides will yield a reconstruction with clearer edges than equidistant projection angles.

In this thesis we will explore the possibilities to choose the projection directions in such a way, that the information gain is maximised. We will discuss a measure for the quality of the image and discuss several methods to determine the choice of projection angles, so that the quality improves maximally with each extra projection.

The structure of the thesis is as follows. In the next section, a short history of CT is given and then the principles behind X-rays will be explained, first the underlying physics and then the mathematics. A mathematical description and the motivation for investigating this particular problem are given in Section 3, followed by the underlying optimisation theory, that is used as an approach to the problem. Afterwards, we will address the difficulties that arise with real data and how we propose to overcome these. In Section 4 the algorithms, used to tackle the problem, will be explained. Then the simulation methods used to test the proposed algorithms, are explained in Section 5. The results of our proposed methods on simulated data are presented in Section 6. Finally we present our conclusions, discussion and suggestions for further research in Section 7. Additional research on noise simulation is discussed in Appendix A. All algorithms designed for this study can be found in Appendices B-E.

2 Background

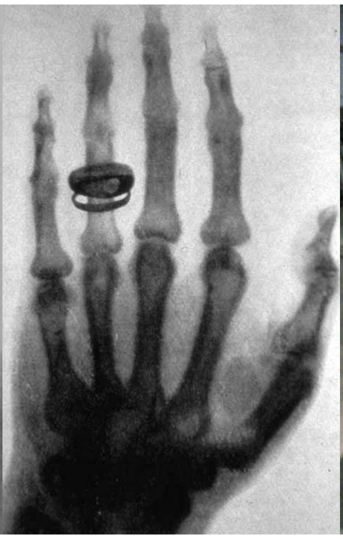
2.1 History

Wilhelm Conrad Röntgen was awarded the Nobel Prize for Physics for discovering X-rays in 1901. He first discovered X-rays in 1895 (he called them X because the rays that produced the image were unknown). One of the first X-ray images he took was one of his wife's hand (see Figure 1(b)). In this picture her wedding ring is clearly visible as the darkest object, as X-rays are blocked by metal [3] [12].

In 1917 the mathematical theory for reconstructing an object from multiple projections was derived by Johann Radon. The *Radon transform* defines the projection of a 2D object by a series of line integrals (see Section 2.3). If for an infinite number of projection directions the transform is known, it is possible to reconstruct the 2D image using the *Inverse Radon transform*. Thus a solution to the *inverse problem* can be found. Due to the computational complexity, however, the application of this solution to X-ray images was not implemented until the 1950's [3] [12].

The foundations of CT were further developed by Allen Cormack. He developed a mathematical theory around image reconstruction in 1956 and experimentally derived the attenuation coefficients of aluminium and wood using his technique. He provided the basis for the implementation of image reconstruction. Godfrey Hounsfield started the development of the first medical CT scanner, as he observed that different attenuation coefficients in the body could be used to reconstruct the internal structure. One of the first medical scanners was installed in 1971 (see Figure 1(c)). In 1979 both Cormack and Hounsfield received the Nobel Prize for their work on Computed Tomography [3] [12].

(a)


(b)

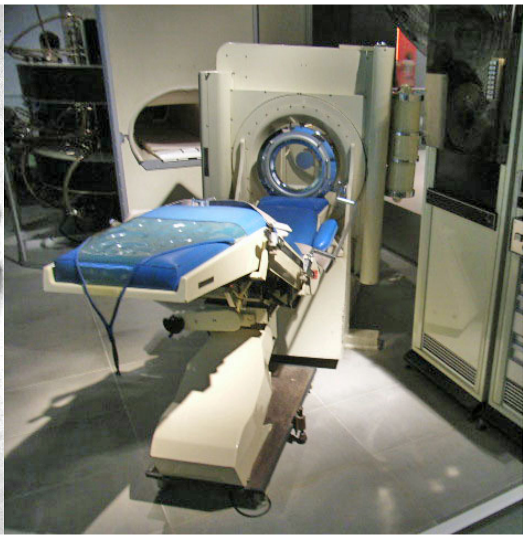

(c)

Figure 1: (a) Wilhelm Conrad Röntgen (1845-1923)[1], (b) X-ray image of the hand of Röntgens wife[2], (c) Early X-ray headscanner [3]

[1]By Nobel Foundation. [Public domain], *Nobel Lectures*, Physics 1901-1921, Elsevier Publishing Company, Amsterdam, 1967
[2]By Wilhelm Röntgen. [Public domain], via Wikimedia Commons
[3]By Philip Cosson. [Public domain], via Wikimedia Commons

2.2 X-ray physics

X-rays are electromagnetic waves with a wavelength between 10^{-8}m and 10^{-13}m. X-ray radiation is typically generated, when electrons are accelerated by applying a high voltage between a cathode and a metal anode. They hit the anode at high speed and are decelerated. During the deceleration process X-ray photons are produced [6].
X-ray photons can interact with the material they traverse. Among other interactions, these are the most important in CT.

- **Photoelectric effect**
  When the energy of the X-ray photon γ is sufficiently high (higher than the binding energy of an electron), the photon can free an electron from an inner shell of the atom (called the photo-electron). The photon is absorbed in this process. An electron from an outer shell then fills the place of the inner electron, while emitting a photon γ' of characteristic radiation. This process happens because the electrons of the outer shells have higher energy than the electrons in inner shells. When an electron moves to a lower shell, it loses energy [6] [12]. See Figure 2.

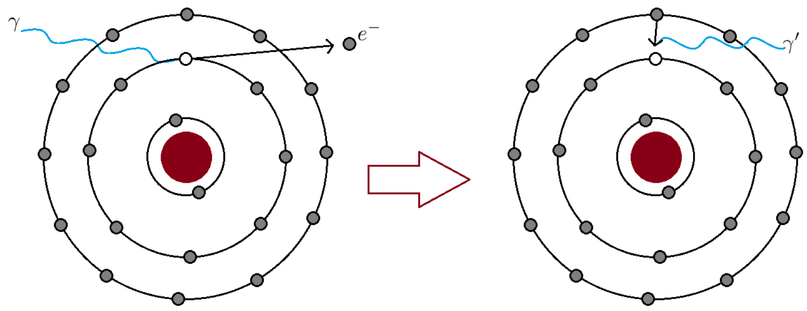


Figure 2: Schematic representation of the photoelectric effect.

- **Compton effect**
  For the Compton effect, the incoming photon also has a higher energy than the binding energy of an outer shell electron. The photon energy is however so much higher, that the photon is not absorbed, but scatters. It continues in a different direction with lower energy, as it loses energy in the collision with the electron (the recoil electron) and part of its energy is transferred to the freed electron [6] [12]. See Figure 3.

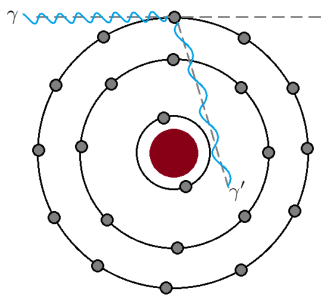


Figure 3: Schematic representation of the Compton effect.

2.3 Data model

When an X-ray beam traverses an object along a line with length Δx, the original intensity I_0 of the X-ray decreases following Lambert-Beer Law:

$$I = I_0 e^{-\mu \Delta x},$$

with μ the linear attenuation coefficient of the material the X-ray travels through, which depends on the scattering interactions of the X-ray with the material. If we view an object as consisting of N slices of thickness Δx each with their own attenuation coefficient μ_i, $i = 1, 2 \ldots, N$, the intensity of the X-ray after travelling through the material becomes

$$I = I_0 e^{-\sum_{i=1}^{N} \mu_i \Delta x}.$$

Rearranging and taking the negative logarithm gives:

$$\sum_{i=1}^{N} \mu_i \Delta x = -\ln(\frac{I}{I_0}).$$

Under certain assumptions, e.g. continuity of $\mu(x)$, the sum becomes an integral when taking the limit $\Delta x \to 0$. This gives the measurement d over an object of length L for a single ray in CT [6], [12]:

$$d = -\ln(\frac{I}{I_0}) = \int_L \mu(x) dx. \tag{1}$$

If the image is 2D, and the source and detector move around the object, the *object function* containing the values of the attenuation coefficient at each location in the object, extends to $\mu(x, y)$. The projections are characterised by the angle θ by which the setup is rotated, see Figure 4. Each projection angle gives projection data $d_\theta(s)$ for each detector offset $s \in \mathbb{R}$. This gives the following formula for the projection data:

$$d_\theta(s) = \int_{L(\theta,s)} \mu(x, y) dl, \tag{2}$$

with $L(\theta, s)$ the line from the source through the object that hits detector offset s, defined by $L(\theta, s) = \{(x, y) \in \mathbb{R} \times \mathbb{R} : x \cos\theta + y \sin\theta = s\}$. Equation 2 is often referred to as the *Radon transform.*

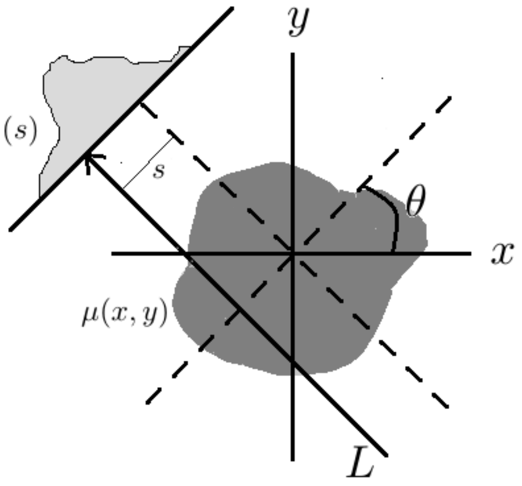


Figure 4: Visualisation of the parameters in Equation 2. Dark grey is the object, light grey the corresponding measurement data.

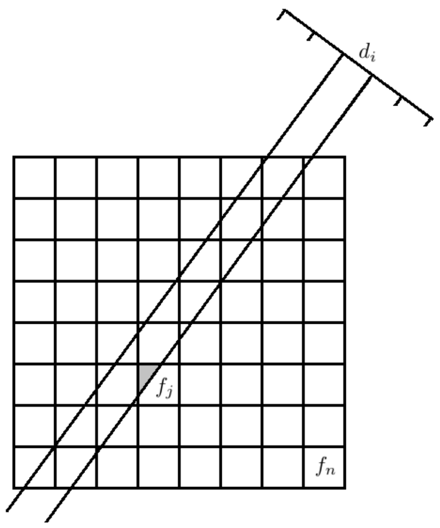


Figure 5: Visualisation of the stripkernel model. Grey area corresponds to the contibution of pixel f_j.

As in real applications it is not possible to take an infinite number of projection angles or an infinite number of infinitely small detector pixels, this equation is used in discretized form. To this end, the image of size $n \times n$ to be reconstructed, is represented as a grid, with pixels f_j and total number of pixels $m = n^2$. The value for detector pixel i can then be represented as the sum of the contributions of each image pixel f_j to the path of the ray from the source to detector pixel i:

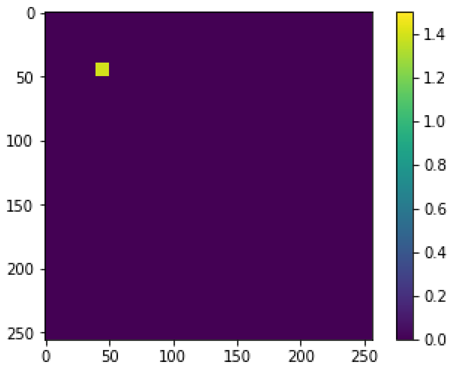
$$d_i = \sum_{j=1}^{n} A_{ij} f_j. \tag{3}$$

In this equation, the coefficient A_{ij} is the weight of pixel f_j. This can vary for example as the ray may have to travel different distances through different pixels. We regard all rays that hit detector pixel i as one *strip*. If the strip covers a large area of pixel f_j, this pixel will contribute more than if the covered area is small. The *strip-kernel* model defines the contribution of each pixel f_j by the area of the intersection of that pixel with the projection strip. This is shown in Figure 5.
The data $\mathbf{d}$ can, following Equation 3 and using vector representations for the images and a matrix representation for the contributions of each pixel, be modelled as follows:

$$\mathbf{d} = \mathbf{A}\mathbf{f}, \tag{4}$$

which is called a *forward projection* of the true image $\mathbf{f}$ by the projection matrix $\mathbf{A}$ [3].

The data $\mathbf{d}$ is often represented as a sinogram, a matrix of size $N_\theta \times N_{detector}$, with N_θ the number of angles and $N_{detector}$ the number of detector pixels. Each measurement is represented by a horizontal line in the sinogram, with the measurements for detector pixel i in column i. When rotating around an object, this means for example that a dot in the object is represented by a sine-shaped line from the top to the bottom of the sinogram. This can be seen in Figure 6.


(a) True image (size 256x256) with dot

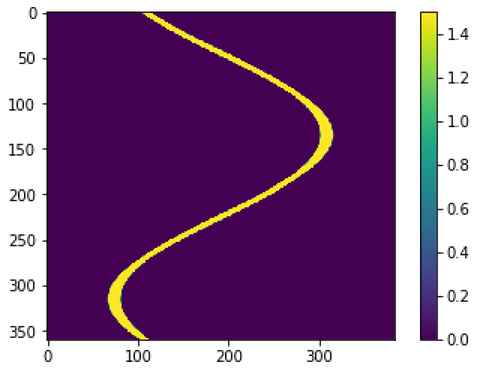

(b) Sinogram of 360 degrees

Figure 6: (a) A square image with a dot and (b) the sinogram of 360 equally spaced projections circling fully around the object once.

2.4 Projection model

The exact form of the data in CT depends on the scanner and detector properties. There are different possible setups of source and detector, depending on beam shape and trajectory of source and detector with respect to the object.

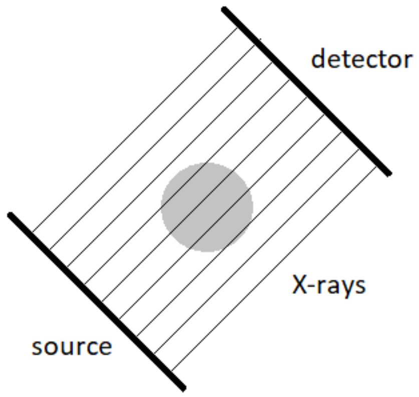


Figure 7: Parallel beam scanner setup.

There are two beam types:

- **Parallel beam**
  All rays are parallel. This is the easiest case to reconstruct and will therefore be mostly used in this thesis, see Figure 7.
- **Cone/Fan beam**
  The source is a point source from which the rays exit in a cone shape. In 2D this becomes a fan shape.

The projection geometry also depends on the trajectory of source and detector. There are two main geometries possible:

- **Circular**
  The source and detector move around the object, in the same plane. This is equivalent to a fixed source and detector with the object rotating around its own axis between them.
- **Helical**
  The source and detector do not only move around the object, but also up and down. Here also different configurations can constitute the same geometry, for example if the object rotates, while the source and detector move in the vertical direction.

3 Theory

Reconstructing images with limited data is one of the problems within Computational Tomography. The available scanning data can be limited due to several reasons. It might, for example, not be possible to fully rotate around an object. It can also happen, that the object is too large to fit its full projection on the detector. A reason for obtaining limited data on purpose is the wish to expose the scanned object to a low radiation dose or limit the scanning time by reducing the number of projections. This is especially important in medical applications as the radiation of X-ray scans increases the risk of the patient developing cancer. This risk depends on the duration and the intensity of the scans [19]. Radiation doses and associated lifetime risks of developing cancer as a result can vary significantly in different CT studies. Age, sex and scan type are factors that influence the risk [23]. The radiation can harm the (biological) sample, so that its properties change and this reduces the quality of the data [11]. This process is called *beam damage*. It also occurs in electron tomography and limits the number of projections that can be taken [17].

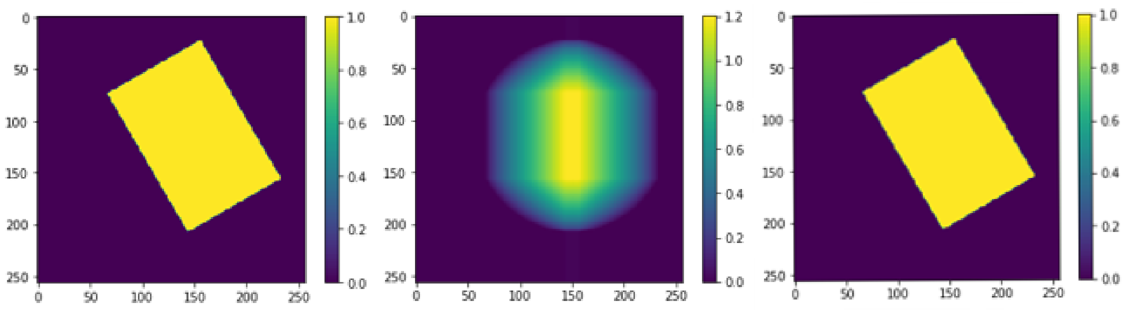
Several methods have been proposed to deal with limited data. For reconstructing a sparse image with under sampled data, methods using compressed sensing were successfully developed [8]. Some algorithms aim to adapt state of the art algebraic, iterative reconstruction algorithms to cope with limited data [24]. Other methods focus on trying to make the data more complete, by adding the missing rows to the sinogram. A method that does this so-called *inpainting* using a total variation minimalisation gives promising results in improving the reconstruction [7].

In many CT applications, the standard way to select angles is to distribute the projection angles equally over the full rotation circle. This means that the angles are chosen equidistantly and often a high number of projections is taken so that these angular intervals are small. An interesting question is whether the same quality of reconstruction can be obtained, using less projections. As for a limited number of projections this is not necessarily the approach that yields the best possible reconstruction, we will in this thesis explore other methods of choosing projection angles.
For selecting the projection angles that give the highest quality reconstruction, one possible research direction is a dynamical approach. The next angle to be scanned is then based on the knowledge gained from the data obtained by previous scans [4]. For this dynamic approach, the new angle has to be calculated quickly in between two projections and is therefore limited by computational power. In this thesis, we will explore static ways to choose a limited number of projection angles, i.e. fixing the projection angles before the scan.

3.1 Problem description

In this thesis, we will discuss the following question: if a maximum number of projection angles is given, can we choose these angles in a such a way, that the quality of the resulting reconstruction is maximised?
To illustrate the importance of choosing the 'right' angles, when facing a limited angle reconstruction problem, we show two reconstructions of a rectangle in Figure 8. On the left is the true image. The reconstructions are made with 100 evaluations of the SIRT algorithm (see Section 5.2). In Figure 8(b) the rectangle is not recognisable in the reconstruction with the two chosen angles. As can be seen in Figure 8(c), it is possible to reconstruct the rectangle from just two projections, if the angles along the sides are chosen. This illustrates the importance of the alignment of the angles with the shape of the object, as both angle sets are equidistant.

One motivating application that we will keep in mind during the process of finding optimal projection angles, is the scanning of wooden shipmodels from the Rijkmuseum in Amsterdam. The art historians specialised in these models are interested in bringing the wood veins and growth rings into view. CT offers a non-destructive method for dating wood, whereas in more conventional dendrochronology a piece of the object has to be cut out to be able to see the rings. Thus, the application of CT to dendrochronology will allow more objects to be dated. To see the year rings, the image resolution and intensity of the X-ray beam are important factors [5]. As the shipmodels of the Rijksmuseum often have the same alignment of wood, we expect the relevant features to be approximately in the same direction for a collection of similar models. Therefore this seems to be an application for which the angle selection methods may be very useful.



(a) (b) (c)

Figure 8: (a) True image: rectangle tilted 30 degrees, (b) Reconstruction using projection angles {0, 90} and 100 iterations of SIRT, (c) Reconstruction using projection angles {30, 120} and 100 iterations of SIRT.

3.2 Optimal experimental design

Finding the 'best' projection angles is an optimisation problem, as we want to minimise the distance between the reconstruction from a limited number of projection angles and the true image $\mathbf{f}$. First we discuss the approach if this true image would be known. Define the set of projection angles $\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots, \theta_{N_\theta}\}$.
The problem of finding the best projection angles is the following optimisation problem:

$$\min_{\boldsymbol{\theta}} \frac{1}{2}||\hat{\mathbf{f}}(\boldsymbol{\theta}) - \mathbf{f}||_2^2. \tag{5}$$

For convenience we will use the notation

$$L(\boldsymbol{\theta}) = \frac{1}{2}||\hat{\mathbf{f}}(\boldsymbol{\theta}) - \mathbf{f}||_2^2. \tag{6}$$

We call $L(\boldsymbol{\theta})$ the *cost function*. This function of the L^2-norm $||x||_2$ is often used in tomography as a measure for the quality of the reconstruction, as it has useful properties for differentiation.
The reconstruction $\hat{\mathbf{f}}(\boldsymbol{\theta})$ that is based on data gained by projections along the angles in the set $\boldsymbol{\theta}$, might not be available explicitly. It is an optimisation problem in itself, as for the reconstruction the goal is to minimise the data misfit. Moreover the data often does not yield a unique reconstruction. The goal in CT is to find the reconstruction, the forward projection of which matches the data best. Another option is to use a function $G(\boldsymbol{\theta})$, which we call the *reconstruction function*, to give an approximation of the reconstruction $\hat{\mathbf{f}}(\boldsymbol{\theta})$ using the data corresponding to the angles in $\boldsymbol{\theta}$. This can be any reconstruction method. In Section 5 we explain how we use the SIRT method for our reconstruction.
During the process, we want to minimise the error of the reconstructed image with respect to the true image. In simulations with self created images, the true image is available and thus we can compare it to the reconstruction. In real life applications the true image is not available and therefore we will need some other way to assess the quality of our reconstruction.

We will introduce theory from a Bayesian framework, which provides a method to approximate the cost function L if the true image $\mathbf{f}$ is not available.

Denote $p(\mathbf{f})$ the probability distribution over images that are likely to be similar to the true image. Assume that the true image $\mathbf{f}$ comes from the same distribution. The cost function L in Equation 6 can then be replaced by the expected Bayes risk, $\mathbb{E}_{p(\mathbf{f})}(\frac{1}{2}||\hat{\mathbf{f}}(\boldsymbol{\theta}) - \mathbf{f}||_2^2)$. This leads to the following minimization problem:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{p(\mathbf{f})}(\frac{1}{2}||\hat{\mathbf{f}}(\boldsymbol{\theta}) - \mathbf{f}||_2^2) = \min_{\boldsymbol{\theta}} \int \frac{1}{2}||\hat{\mathbf{f}}(\boldsymbol{\theta}) - \mathbf{f}||_2^2 p(\mathbf{f}) d\mathbf{f}. \tag{7}$$

We assume that we have training samples $\mathbf{t}_i \sim p(\mathbf{f})$ for $i = 1, 2, \ldots, N_{train}$, of which the true image $\mathbf{t}_i$ is known. Then the cost function L (Equation 6) can be approximated by the empirical Bayes risk:

$$L_{bayes} = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} \frac{1}{2}||\hat{\mathbf{t}}_i(\boldsymbol{\theta}) - \mathbf{t}_i||_2^2. \tag{8}$$

The set of training samples is called the *a-priori choice* and L_{bayes} the *surrogate cost function*. This approach incorporates prior knowledge that we might have of our object, in the form of its probability distribution. Therefore we expect that choosing the projection angles based on this will be closer to the 'best' angles than equidistantly chosen projections.
We would like to take one step further towards dynamical angle selection, in which the choice of the next projection angle is based on the data from previous projections. In our simulations, we investigate the cost function and the minimum that defines which angles should be chosen. This is done for a cost function without prior knowledge and for a cost function in which the information from the vertical angle, i.e. the angle along the y-axis, is taken into account as if the data from that projection were already known.

4 Methods: Angle selection

In this section we describe algorithms designed to find an optimal set of projection angles. All algorithms have been programmed and tested using Python and can be found in the Appendix. In this thesis, three algorithms will be discussed: a greedy algorithm, a coordinate descent algorithm and an ensemble Kalman Filter algorithm. Each of these are first described mathematically and subsequently we briefly discuss the implementation. In the last part of this chapter we discuss how to adjust the algorithms, so that they can be used with either the true image or a training set of true images.

4.1 The greedy algorithm

As a first approach to the angle selection problem, we have developed a greedy algorithm. This algorithm iteratively adds the angle that gives lowest evaluation of the cost function L to the set of chosen angles. See Algorithm 1.

To find the angle set, a first angle is pre-specified. If possible based on the prior knowledge of the true image. This is because we have seen that choosing the first angle manually, improves the algorithm a lot, as reconstructions from just one projection do not give enough information to let the algorithm choose the right starting angle. It is possible to specify the first angle, as we assume that in real life the alignment of the object in the scanner is known. Thus for a rectangular object for example, it is possible to make sure that the first angle is along the long side. As we can choose the position of the object in the scanner, this is a simple way to incorporate prior knowledge of the shape of the object to improve angle selection. The number of angles desired needs to be specified as input of the algorithm.

Algorithm 1: Greedy

Input:

- θ_0, the first angle, based on the alignment of the object in the scanner. If none is given, a random first angle will be chosen;
- N_θ, the desired total number of angles in the set $\boldsymbol{\theta}$;
- $L(\boldsymbol{\theta})$, the cost function.

Result: A set $\boldsymbol{\theta}$ of N_θ projection angles

```
Initialization; $\boldsymbol{\theta} = \{\theta_0\}$
for $j \in 1, 2, \ldots, N_\theta$ do
    $\theta_j = \text{argmin}_{\theta^* \in [0,\pi)} L(\boldsymbol{\theta} \cup \theta^*)$
    $\boldsymbol{\theta} = \boldsymbol{\theta} \cup \theta_j$
end
```

4.1.1 Implementation

To find the angle θ^* that should be added to the current angle set, we use a function that minimises $L(\boldsymbol{\theta} \cup \theta^*)$ for θ^* (see Appendix C). The cost function L is first evaluated on a grid with step size one degree within a given range. For cost functions with many local minima, a finer grid may be needed for the algorithm to find the global minimum. The grid ensures that we find the global minimum at least to the precision of the grid. The function then performs a local search near this minimum using the minimiser *fmin* of the Python *scipy* package to improve the minimum found on the grid.

The range for the greedy algorithm is chosen to be $[0, \pi)$, because in a parallel beam setup with noiseless measurements the projection θ and $\theta + \pi$ yield the same data [3]. This is because the density of an object is the same along a line, regardless of the direction along which the X-ray beam travels through the material. Therefore the same fraction of the intensity of the beam is blocked.

4.2 The coordinate descent algorithm

To improve on any chosen set of projection angles, for example the output of the greedy algorithm, we use a coordinate descent method. The algorithm takes a set of input angles and changes one angle at a time, keeping the others fixed. It changes the current angle to a better position between the angles to the left and right of it. Thus it is shifted, but not further than its neighbours, see Algorithm 2. Note that with each update the angle can only move to a position with lower cost function value, as the current angle is always in the set of possibilities and can therefore be chosen, if it takes the minimum value on the interval. The initial set of possible angles can for example be a set of angles that is acquired using the greedy algorithm. If no initial set of angles is given, the function will take equidistant angles between 0 and π.

Algorithm 2: Coordinate Descent

Input:

- $\boldsymbol{\theta}^0$, the initial guess for the set of angles, if none is given, the initial guess will be equidistant angles;
- n_iter, the number of iterations;
- $L(\boldsymbol{\theta})$, the cost function;
- (optional, if no initial set is given) N_θ, the desired total number of angles in the set $\boldsymbol{\theta}$, input required to make an initial set of N_θ equidistant angles.

Result: A set $\boldsymbol{\theta}$ of N_θ projection angles

```
Initialization; $\boldsymbol{\theta} = \boldsymbol{\theta}^0$
for $k \in 1, 2, \ldots, n\_iter$ do
    for $i \in 1, 2, \ldots N_\theta$ do
        $\boldsymbol{\theta} = \boldsymbol{\theta} \setminus \theta_i$
        $\theta_i = \min_{\theta^* \in [\theta_{i-1}, \theta_{i+1})} L(\boldsymbol{\theta} \cup \theta^*)$
        $\boldsymbol{\theta} = \boldsymbol{\theta} \cup \theta_i$
    end
end
```

4.2.1 Implementation

Like the greedy algorithm, the coordinate descent algorithm uses the function, described in Section 4.1.1, to find a minimum of the cost function on an interval. The interval is between the angles left and right of the current angle to be updated. An angle giving a lower or equal cost function value with respect to the current angle is found and the current angle is updated to that position. This is repeated for all the angles in an initial angle set in turn for a chosen number of iterations. The total number of angles does not change during the coordinate descent algorithm.

4.3 The ensemble Kalman algorithm

The algorithms described above look for a local minimum of the cost function. In this section we will introduce an algorithm that searches for the global minimum and will thus be better equipped to handle cost functions with multiple local minima.

The ensemble Kalman method for finding a minimum updates an initial ensemble (or set) of particles [15]. This update is based on the evaluations of the cost function that are already known. In our case the particles each are a set of a fixed number of projection angles. For example in the case of two angles a particle could be $\{30, 120\}$. The set of particles is called the *ensemble*, denoted by $\mathbf{E} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{n_E}\}$, containing a total of n_E particles. Each particle contains N_θ angles, i.e. $\mathbf{p}_l = \{\theta_1^l, \theta_2^l\}$ in the case of two angles.

In each iteration of the algorithm, one or more of these particles are updated, by adding a linear combination of the other particles. This update step is based on a Gaussian approximation. As the number of particles

grows larger, the estimate from the ensemble Kalman filter corresponds to the best linear estimator of the mean conditioned on the known data [22]. In practice this means, that the particles are updated in the direction of the particles that have a lower cost function evaluation.
The updated particles stay in the span of the original particles [15]. The size of each particle stays the same during the iterations of the algorithm. This means, that the number of projection angles to be chosen must be specified beforehand. Like our previous methods, no derivative of the cost function is needed.
An initial ensemble of particles is given to the algorithm. In most cases this ensemble will be randomly generated. The ensemble is stored as a matrix, the particles are columns containing the projection angles. The total number of particles is the number of columns in the initial ensemble. After the update iterations have finished, the algorithm evaluates the cost function for all final particles and outputs the particle (angle set) with lowest cost function evaluation.
In each iteration the algorithm computes for the ensemble $\mathbf{E}$, consisting of n_E particles $\mathbf{p}_l$ and reconstruction function G:

- The ensemble mean (a vector with the means of all first, second, ... angles)

$$\bar{E} = \frac{1}{n_E - 1} \sum_{l=1}^{n_E} \mathbf{p}_l. \tag{9}$$

- The mean of the evaluation of the cost function G

$$\bar{GE} = \frac{1}{n_E - 1} \sum_{l=1}^{n_E} G(\mathbf{p}_l). \tag{10}$$

- The covariance matrix associated with ensemble mean

$$C^{uw} = \frac{1}{n_E - 1} (G(\mathbf{E}) - \bar{GE}) \times (\mathbf{E} - \bar{E}). \tag{11}$$

- The covariance matrix associated with the evaluation mean

$$C^{ww} = \frac{1}{n_E - 1} (G(\mathbf{E}) - \bar{GE}) \times (G(\mathbf{E}) - \bar{GE}). \tag{12}$$

These formulae are used for updates of the ensemble, following [15] (see Algorithm 3).

Algorithm 3: Ensemble Kalman

```
Input:
  • $\mathbf{E}^0$, the initial ensemble, with particles size $N_\theta$;
    and total number of particles $n_E$;
  • $n_{iter}$, the number of iterations of the ensemble Kalman algorithm;
  • $n_{update}$, the number of particles to be updated per iteration;
  • $\mathbf{s}$, an array of the stepsize in each iteration, of length $n_{iter}$ ;
  • $G(\boldsymbol{\theta})$, a function which outputs a reconstruction;
Result: A set $\boldsymbol{\theta}$ of $N_\theta$ projection angles

Initialization; $\mathbf{E} = \mathbf{E}^0$
for $j \in 1, 2, \ldots, n_{iter}$ do
    Calculate:
      • $\bar{E}$ (Equation 9)
      • $C^{uw}$ (Equation 11)
      • $\bar{GE}$ (Equation 10), while storing the value $G(\mathbf{p}_l)$ of each particle)
      • $C^{ww}$ (Equation 12)
    Sort the particles from highest to lowest cost function evaluation $G(\mathbf{p}_l)$
    for $i \in 1, 2 \ldots, n_{update}$ do
        Update $\mathbf{p}_i$ by
            $\mathbf{p}_i^j = \mathbf{p}_i^{j-1} + \mathbf{C}^{uw}(\mathbf{C}^{ww} + \boldsymbol{\Gamma})^{-1}(\mathbf{f} - G(\mathbf{p}_i^{j-1}))$    (13)
        with $\boldsymbol{\Gamma}$ a diagonal matrix with the stepsize $\mathbf{s}[j]$ on the diagonal.
        For the other particles: $\mathbf{p}_i^j = \mathbf{p}_i^{j-1}$
        $\mathbf{E}^j = \bigcup_{i=1}^{n_E} \mathbf{p}_i$
    end
end
$\boldsymbol{\theta} = argmin_{\mathbf{p}_i} G(\mathbf{p}_i)$
```

4.3.1 Implementation

The first adaptation of the original filter is that the number of particles to be updated in each iteration of the algorithm can be chosen. To choose which two particles should be updated, the particles are sorted from high to low cost value and only the two particles with the highest cost function are updated. A difference for implementation with respect to the greedy method and the coordinate descent method, is that the ensemble Kalman method needs the explicit reconstruction $\hat{\mathbf{f}}$ given by $G(\boldsymbol{\theta})$, not just the cost function L. This is because the update step assumes that the cost function is the cost function as described in Equation 6. Therefore, it is not easy to use other cost functions within the algorithm, although the reconstruction method can still be chosen freely.

A second adaptation is that in the update step, a step size can be incorporated. The step size determines the difference between the original position of the particle and its next position. If this step size is large, the particle may be updated too far and not get closer to the minimum, if it is too small, the algorithm takes long to converge. This varies for different objects. The step size can be given as input as a vector of the length of the number of iterations, containing the step size for each iteration. For our simulations we choose step size 1 for each iteration, which is the same as the original ensemble Kalman method.

After each iteration the angles are taken modulo π to make sure they lie in $[0, \pi)$, and subsequently they are sorted. This is necessary, as the update step may cause one of the angles of a particle to be outside the triangle

defined by $\{(\theta_1, \theta_2) : \theta_1, \theta_2 \in [0, \pi), \theta_1 \leq \theta_2\}$ in the case of two angles. If the algorithm has enough iterations, all particles end up in the same (local) minimum. Whether this is also the global minimum, depends on the initial set of particles.
In the update step (the second for-loop) of the algorithm, we need to invert $C^{ww} + \Gamma$. As these are large matrices which is computationally inefficient, we use the Woodbury matrix identity. [21]
The Woodbury matrix identity is given by:

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}. \tag{14}$$

We introduce the following two matrices:

$$J_{uw} = \frac{1}{\sqrt{nE - 1}}(E - \bar{E})$$

$$J_{ww} = \frac{1}{\sqrt{nE - 1}}(G(E) - \bar{GE})$$

It follows that:

$$C^{uw} = J_{uw}J_{ww}^T$$

and

$$C^{ww} = J_{ww}J_{ww}^T.$$

We then use Equation 14, with $A = \Gamma$, $C = Id$, $U = J_{ww}$ and $V = J_w w^T$ to get

$$C^{ww} + \Gamma = \Gamma^{-1} - \Gamma^{-1}J_{ww}(Id + J_w w^T \Gamma^{-1} J_{ww})^{-1} J_w w^T \Gamma^{-1}.$$

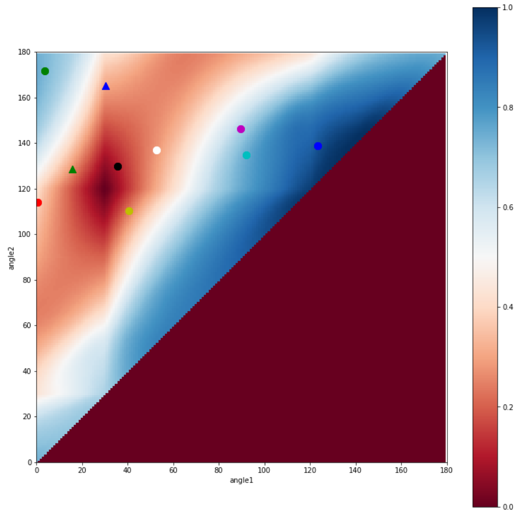
The matrix to be inverted $(Id + J_{ww}^T \Gamma^{-1} J_{ww})$ now has size $n_E \times n_E$ and Γ is the identity times a step size, so its inverse is the identity matrix times one over the step size. This inversion is computationally easier than the original inversion.
For illustration purposes, the the propagation of the algorithm for a rectangle tilted 30 degrees is shown in Figure 9. The cost function for two angles is shown in the background. The starting ensemble consists of 10 randomly drawn particles that each contain 2 angles. These are represented by a dot, for which the first angle corresponds to the x-axis and the second to the y-axis. Per iteration a fixed number of particles is updated, in this case 2. It can be seen that the particles that are furthest away from the minimum are updated first and that for this simple phantom all the particles end up in the global minimum in (less than) 10 iterations. There is no guarantee that the algorithm converges to the global minimum.

4.4 Training data

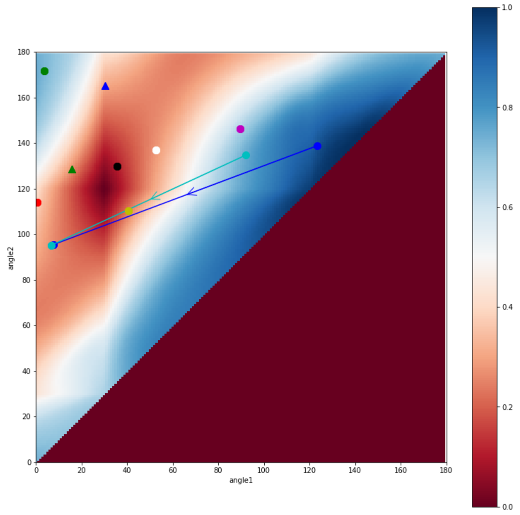
When we are using likely training images to approximate the cost function of the true image, the algorithms will have to be slightly modified. Instead of having the true image of our object to compare the reconstruction with, there will be several true images from the training set and corresponding reconstructions.
We define the set of true images $\mathbf{T}$ and the set of reconstructions $\hat{\mathbf{T}}(\boldsymbol{\theta})$:

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 \\ \mathbf{t}_2 \\ \vdots \\ \mathbf{t}_{N_{train}} \end{bmatrix} ; \quad \hat{\mathbf{T}}(\boldsymbol{\theta}) = \begin{bmatrix} \hat{\mathbf{t}}_1(\boldsymbol{\theta}) \\ \hat{\mathbf{t}}_2(\boldsymbol{\theta}) \\ \vdots \\ \hat{\mathbf{t}}_{N_{train}}(\boldsymbol{\theta}) \end{bmatrix}$$

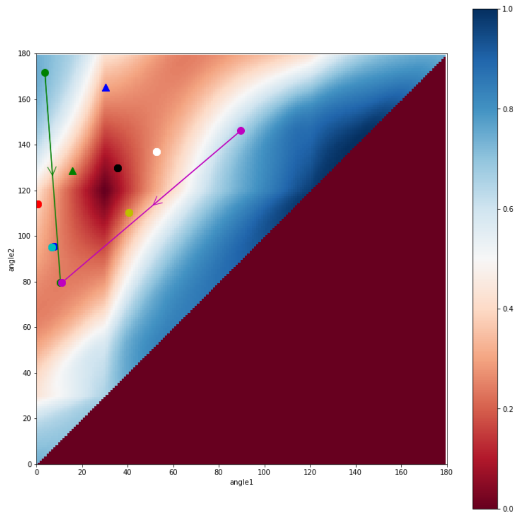
We use these as input for our cost- and reconstruction functions. Note that using $\mathbf{T}$ and $\hat{\mathbf{T}}$, it is also possible to use the same algorithms for the case where the true image is available, by defining $\mathbf{T} = \mathbf{f}$. The true image set then consists of just one image. We have therefore implemented this in such a way, that the algorithms can be used for both situations that the true image is available and in which we use training samples.



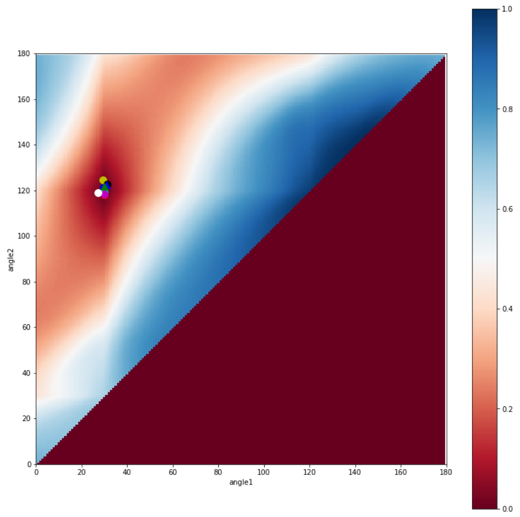
(a) Starting ensemble



(b) The particle updates in the first iteration



(c) The particle updates in the second iteration



(d) The ensemble after 10 iterations

Figure 9: The cost function L for a rectangle tilted 30 degrees and the ensemble Kalman algorithm with 10 particles, two particles updated in each iteration with stepsize 1.

5 Methods: Simulation

For testing the algorithms described in the previous section, simulations have been designed. These are all programmed in Python and can be found in the Appendix. Furthermore, to compute forward projections and reconstructions the Astra Toolbox for Python was used [1] [2].

5.1 Phantoms

A test image whose properties are known, is called a 'phantom'. A phantom is created by defining the values of a matrix. We have created a few simple functions that generate phantoms on which we can test our optimisation programs. To begin with, we use 2D phantoms. A generalisation to 3D can be made in further investigations. As the object in the scanner rotates only around its own axis and does not move in the vertical direction, the set of possible projection angles is the same as for 2D objects. It is possible to extend the simulations to 3D objects, using the Astra Toolbox.

The phantoms, used in this thesis, consist of a rectangle, a circle and a simple image that has a shape similar to a horizontal slice through a ship (Figure 10). With some imagination, the boat shape and the rowing benches can be seen. All phantoms can be resized to any square size and tilted to make sure they are not aligned with the rows and columns of the matrix describing the phantom. See Appendix B for the exact definition of each of these functions. In Section 6.3, we moreover explain the use of a golden standard reconstruction, gained from data from the FleX-ray scanner at CWI, can be used as a phantom.

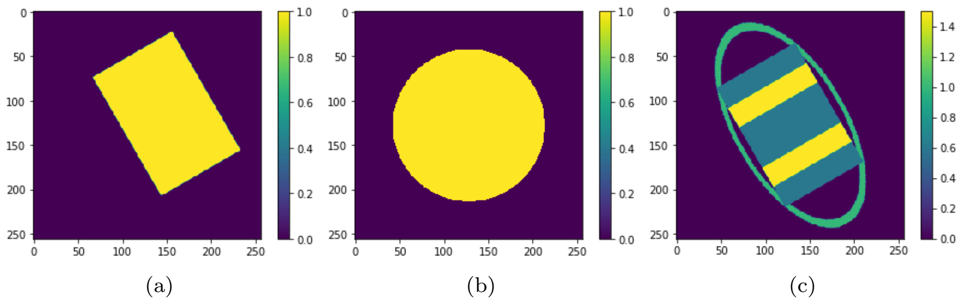


Figure 10: The three phantoms: (a) rectangle, (b) circle and (c) boat.

5.2 Reconstruction function G and cost function L

As mentioned before in Section 1, the data can be modelled as follows:

$$\mathbf{d} = \mathbf{A}\mathbf{f}. \tag{15}$$

The forward operator $\mathbf{A}$ depends on the configuration of the X-ray scanner (i.e. beam type and projection geometry) and the projection angles. The forward operator is applied to the phantom image to create data. The projection geometry used in the simulations is parallel beam. In our simulations we use noiseless data. How to simulate noise is discussed in the section with additional material, Section A.
For our purpose, we will use the function L (see Algorithm 6) as the cost function for the optimisation problem described in Equation 5. This uses an approximation of the reconstruction, given by G (Algorithm 5). For G, we chose to use the SIRT algorithm (algorithm 4) to create a reconstruction $\hat{\mathbf{f}}$. We can use a low number of iterations, as we do not need to have a good reconstruction within the optimisation algorithm, merely a reconstruction that is good enough to distinguish which angles are better than others. For our phantoms, we have experimentally seen that performing 5 iterations already yields a good reconstruction. For example for the rectangle the angles along the sides are found. For computational reasons we therefore use 5 iterations for the SIRT algorithm, unless otherwise mentioned. More iterations would improve the methods and may be necessary when the phantoms are more complicated. This can be changed, as the number of internal SIRT iterations is an input parameter of the algorithms.

The SIRT algorithm that is used within G, is an iterative algorithm that updates the reconstruction in each step by adding a weighted back projection of the difference between the data and the forward projection of the current reconstruction. One iteration of the SIRT algorithm is described in Algorithm 4 [3]. For the unconstrained case, the SIRT algorithm converges to the weighted least squares solution $\hat{\mathbf{f}} = \text{argmin}_{\mathbf{f}^* \in \mathbb{R}^n} ||\mathbf{A}\mathbf{f}^* - \mathbf{d}||_{\mathbf{R}}^2$ [14]. During the algorithm, non-negativity of the reconstruction is enforced, as the density of objects is always non-negative.

Algorithm 4: SIRT iteration

1. Compute the forward projection of the current reconstruction: $\mathbf{d}^i = \mathbf{A} * \hat{\mathbf{f}}^i$.
2. Compute the *projection difference* of this reconstruction: $\mathbf{r}^i = \mathbf{d} - \mathbf{d}^i$.
3. Compute $\mathbf{R}$ and $\mathbf{C}$. The diagonal matrix $\mathbf{R}$ consists of the inverse row sums of $\mathbf{A}$: $R_{ii} = \frac{1}{\sum_j A_{ij}}$, $R_{ij} = 0$ if $i \neq j$. $\mathbf{C}$ is a diagonal matrix of the inverse column sums of $\mathbf{A}$: $C_{ii} = \frac{1}{\sum_i A_{ij}}$, $C_{ij} = 0$ if $i \neq j$.
4. Update the data:
$$\mathbf{d}^{i+1} = \mathbf{d}^i + \mathbf{C}\mathbf{A}^T\mathbf{R}\mathbf{r}^i.$$
In this update step, a weighted version of the backprojection of the projection difference is added to the current data.
5. Set all negative values of $\mathbf{d}^{i+1}$ to 0 (non-negativity constraint).

Algorithm 5: The reconstruction function G

Input: Projection angles $\boldsymbol{\theta}$, *n_iter_sirt*
Result: A reconstruction based on projection angles $\boldsymbol{\theta}$

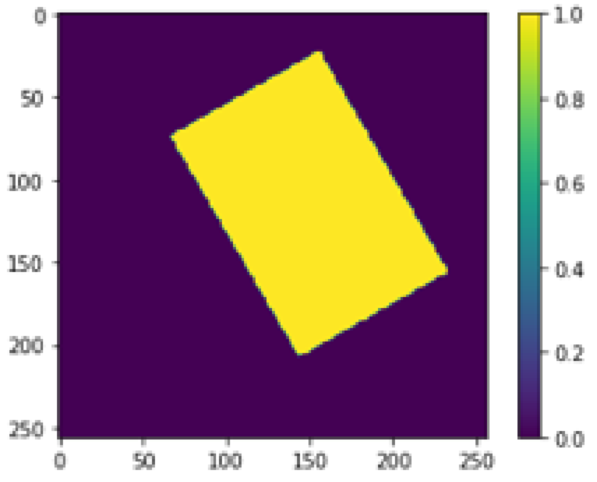
1. Do a forward projection of $\mathbf{f}$ using the projections corresponding to $\boldsymbol{\theta}$, to create $\mathbf{d}$
2. Make a reconstruction, $\hat{\mathbf{f}}$, of $\mathbf{d}$ with *n_iter_sirt* iterations of the SIRT algorithm (Algorithm 4), imposing non-negativity.

Algorithm 6: The cost function L

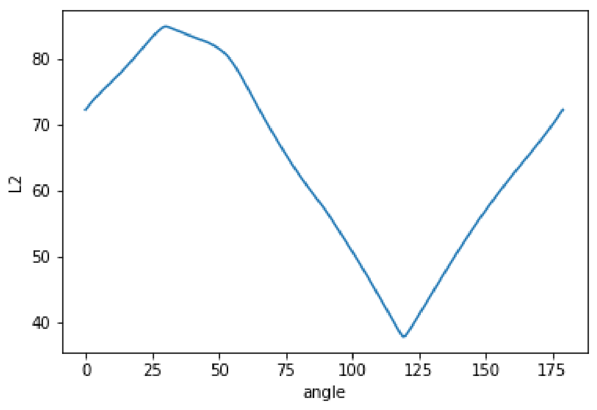
Input: Projection angles $\boldsymbol{\theta}$
Result: The cost function value of a reconstruction based on projection angles $\boldsymbol{\theta}$

1. Calculate $\hat{\mathbf{f}} = G(\boldsymbol{\theta})$
2. Calculate $L = \frac{1}{2}||\hat{\mathbf{f}}(\boldsymbol{\theta}) - \mathbf{f}||_2^2$

In Figure 11 the cost function L for a reconstruction using SIRT for a rectangle tilted 30 degrees is shown for adding one angle to the pre-specified first angle (in this case 30 degrees). It can be seen that there is a local minimum around 120 degrees.
The cost function for the same tilted rectangle for two angles with the first and second angle on the x and y axis, is shown in Figure 12. We would expect the cost function to be mirrored in the diagonal, as for the reconstruction the order of the angles does not matter. Because of this mirror effect, the angles have to be ordered to be able to optimise over several angles simultaneously. If both configurations are taken into account there are two global minima, which makes finding the minimum more difficult. The values are lowest on the diagonal, as scanning the same angle twice does not improve the reconstruction. In Figure 12 indeed a minimum exists at (30,120) and (120,30) and the cost function is mirrored in the diagonal. As the rectangle is tilted to the left by 30 degrees, these angles correspond to the scanning directions along the sides of the rectangle.

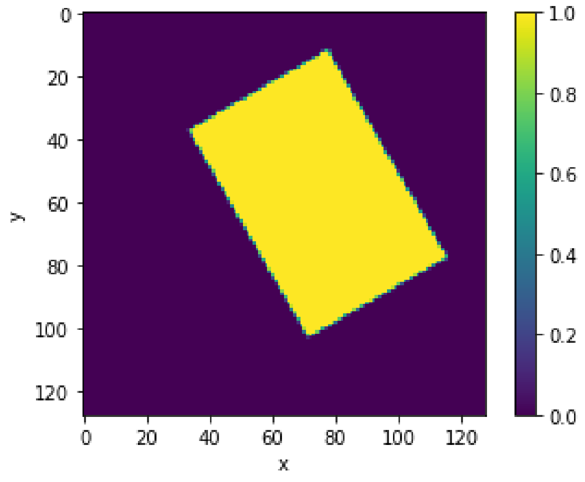


(a) True image

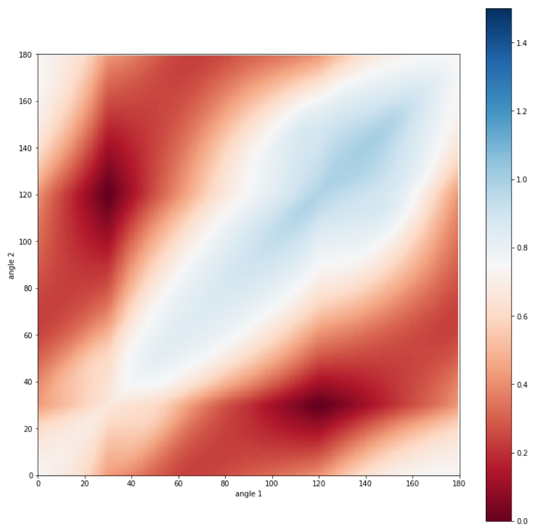


(b) Cost function L, for a reconstruction with 2 angles

Figure 11: L cost function evaluation of an image of a rectangle (image size 256) tilted 30 degrees. The picture shows the cost function value of a reconstruction using SIRT of two angles: first (fixed) angle (30 degrees) and the angle on the x-axis.



(a) True image (128x128 pixels)



(b) Cost function for a reconstruction with 2 angles

Figure 12: A 30 degree tilted rectangle and the corresponding cost function for two angles, scaled.

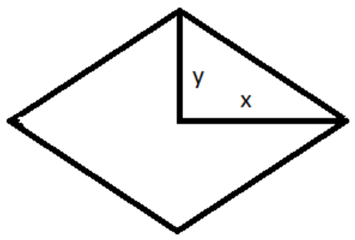
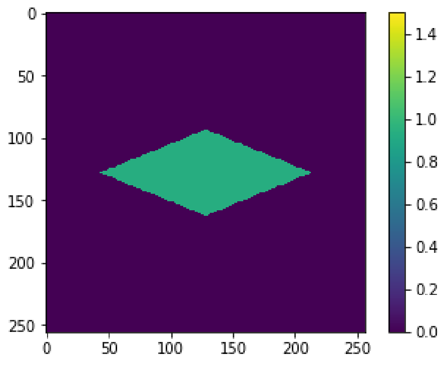


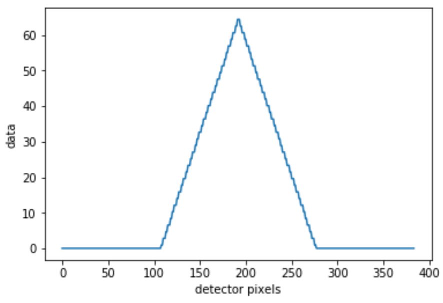
Figure 13: The axes x and y of the diamond.

5.2.1 Training samples

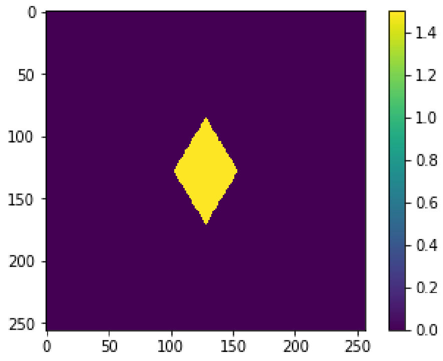
To test the theory on using training samples if the true image is not available, outlined in Section 3.2, a training set and true image from the same probability distribution are needed. To simulate this, a *generator* of training samples was designed. The generator produces a number of images that consist of diamond shapes (see Appendix B.2). The center of these diamonds is always the center of the image, but the length of the horizontal and vertical axis is variable. A variation of 10% in the vertical direction means, that vertical axis y (see Figure 13) is calculated as $y = \bar{y} + \alpha\bar{y}$, with $\bar{y}$ the mean length of the vertical axis and α a randomly drawn number between -0.1 and 0.1. In the same way the x-axis is modified with a randomly drawn number β: $x = \bar{x} + \beta\bar{x}$. The total density of the diamonds from one call of the generator function is the same. When the vertical projection is taken, it is clear what the width of the diamond is. Two diamonds with variation in both directions and two with no variation in the horizontal direction are shown in Figure 14, together with their data when only scanning the zero angle. Because the density is the same for each diamond, the height of the diamond is still uncertain. In this way, we can test whether the prediction of the cost function based on training samples, is better when the additional information from the first scan is added to the prior knowledge. To test this, we can easily create diamonds that have the same width as is measured from the first scan.
From these training samples we draw one 'true' image, that has the mean width and height that is used as input of the generator. To test whether the average cost function of the samples resembles the cost function of this image, we will compute the cost function of different sets of training samples: a set with only the prior knowledge that the true image comes from the same distribution as the training samples, and a set where the vertical angle has been scanned as thus the width of the true image is known.



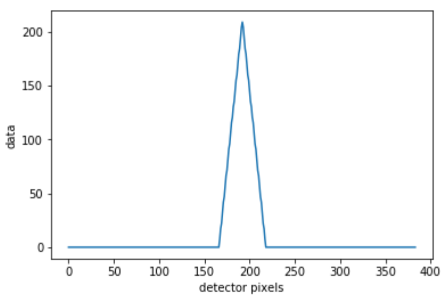
(a)



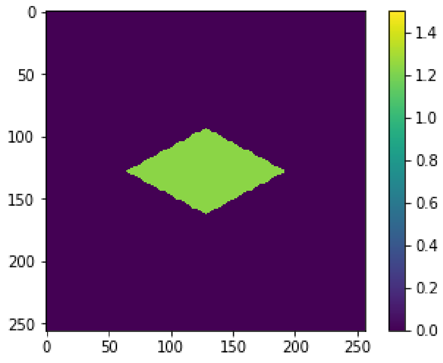
(b)



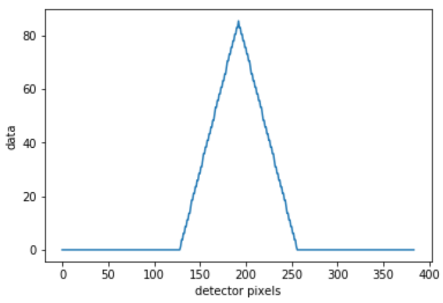
(c)



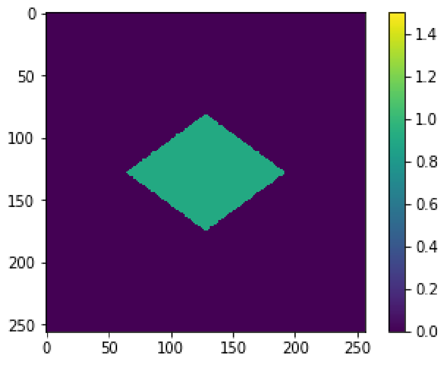
(d)



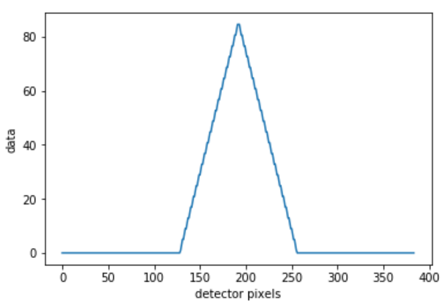
(e)



(f)



(g)



(h)

Figure 14: Two diamonds from the generator and the data the projection of the vertical angle (y-direction produces. (a)-(d) 10% variation in vertical direction, 30% in horizontal direction. (e)-(f) 10% variation in vertical direction, no variation in horizontal direction.

6 Results

6.1 Angle selection algorithms

For testing the algorithms described in Section 4, three phantoms were designed. In this section we will discuss the results of our algorithms selecting two and ten angles respectively and compare the performance to equidistant angle selection. After that we will show an example using a golden standard reconstruction made from real data, as phantom. We will again discuss the performance of the algorithms on this phantom and compare with equidistant angle selection.

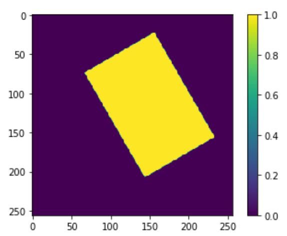
6.1.1 Reconstruction with 2 projection angles

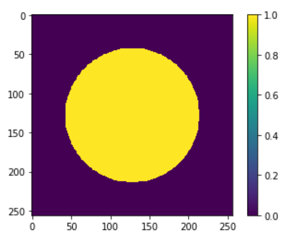
In Figure 15 the phantoms are shown, together with the results for reconstructions with angles chosen by the three algorithms. The results for choosing two angles with the greedy algorithm, are shown in Figures 15 (d)-(f). The angles chosen by the algorithm are shown to the right of the reconstruction. For the greedy algorithm, the first angle, θ_0, is chosen to be aligned with the tilt of the shapes (30 degrees). Therefore, the algorithm only chooses the second angle. The rectangular shape is reconstructed almost perfectly. The circle and the boat cannot be reconstructed as well as the rectangle from only two angles, as more information is needed to reconstruct a bend shape. For the boat the rectangular shapes are clearly visible in the reconstruction.
The greedy method is not an optimal way of choosing the projection angles. We will illustrate this with an example, shown in Figure 16. Imagine scanning a uniform 2D circular object and using the greedy method to select the three angles that give the lowest cost function (Equation 6) for a reconstruction with three projection angles. The greedy algorithm will first select two angles perpendicular to each other and then one other angle (Figure 16(b)). For a uniformly distributed circular 2D object, a reconstruction with a lower cost function value can be obtained using three equidistant angles, i.e.$\{0, 60, 120\}$ (Figure 16(c)). The greedy algorithm is however easy to implement and can be useful for a quick initial guess for the angles. The performance also depends on the choice of the first angle.

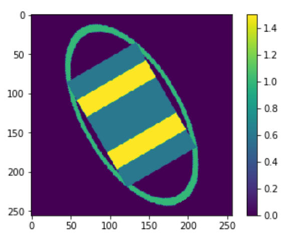
The angles found by the greedy algorithm, are used as the initial angles for the coordinate descent method. The number of iterations of the coordinate descent algorithm to improve the angle set is 10. The results are shown in Figure 15 (g)-(i).
As can be seen, the algorithm improves the cost function evaluation only for the circle. This can be explained by the observation that the angles chosen by the greedy algorithm give a reconstruction that cannot be improved unless more projections are added.

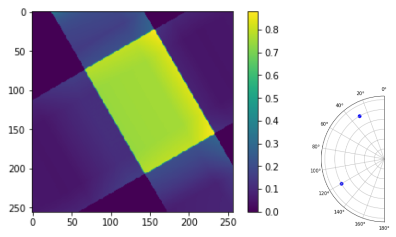
If the image is tilted by 30 degrees (the angles that give lowest cost function are then $\{30, 120\}$) and we give the coordinate descent algorithm initial angles $\{0, 90\}$, the coordinate descent algorithm still finds the optimal angles 30 and 120 in a few iterations. The greedy algorithm will retain the 0, because of our assumption that the first angle is well chosen. The greedy method can thus not produce the optimal same angle set, if the first angle is not chosen optimally. Therefore it may be useful to use the coordinate descent algorithm after the greedy algorithm to improve on the angle set. In that way we compensate for (slight) misalignment of the first angle of the greedy algorithm with the shape of the object.

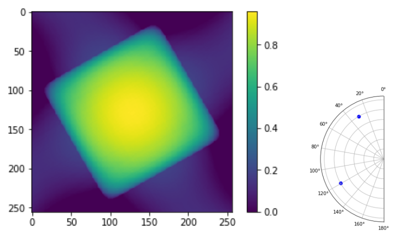
For the results on the ensemble Kalman method, the two particles with highest L2-norm are updated in each iteration and the number of iterations is 10. The 10 initial particles are chosen randomly, a random seed is set to ensure the same results on repetition. The ensemble Kalman method is slightly off in finding the angles for the rectangle, leading to a higher cost function evaluation, but this is in the same order of magnitude as the evaluation of the reconstruction using the angles found by the other methods.

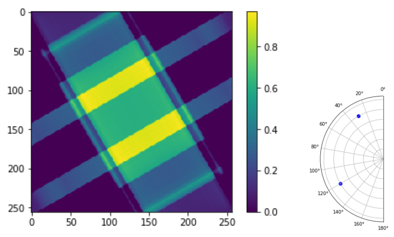

(a) True

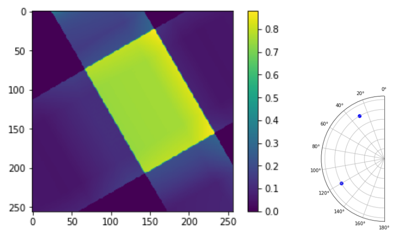

(b) True

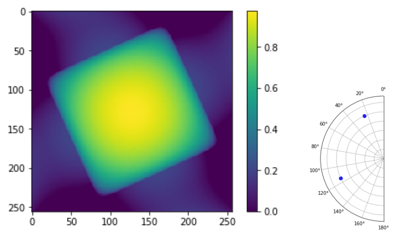

(c) True

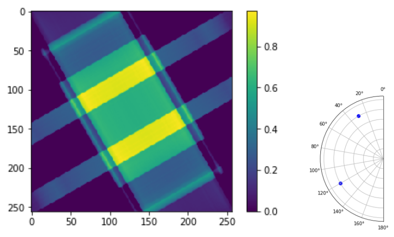

(d) Greedy, L(30,120) = 18.92

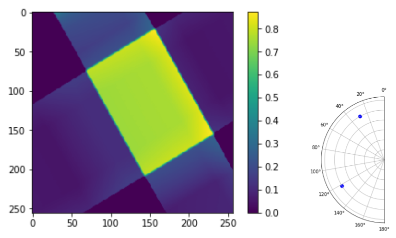

(e) Greedy, L(30,119) = 27.11

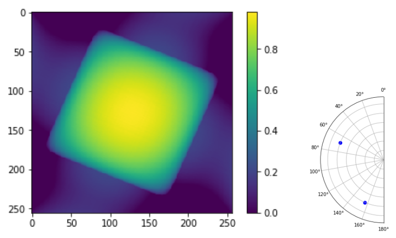

(f) Greedy, L(30,120) = 34.12


(g) Coordinate, L(30,120) = 18.92


(h) Coordinate, L(30,119) =27.11


(i) Coordinate, L(30,120) = 34.12


(j) Ensemble, L(29,121) = 19.63


(k) Ensemble, L(68,156) = 27.09

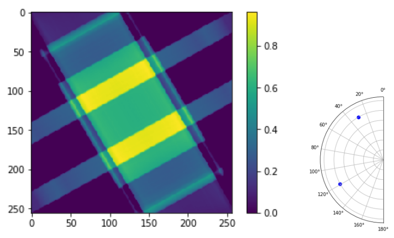

(l) Ensemble, L(30,119) = 34.23

Figure 15: Results of the angle selection methods for two angles. Top: true images, below: reconstructions using the angles found by the methods, chosen angles shown to the right of the reconstruction, corresponding cost function value L given in the captions.

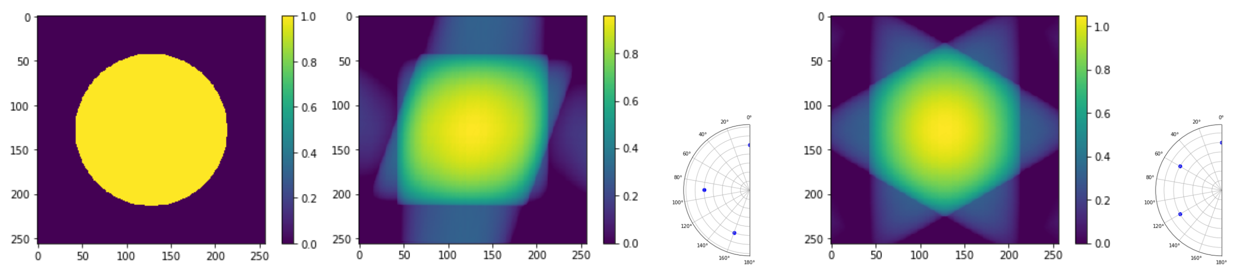

(a) True image (b) Greedy, $L(\{0, 89, 160\}) = 58.21$ (c) Equidistant, $L(\{0, 60, 120\}) = 52.11$

Figure 16: Illustration of the sub-optimality of the greedy angle selection method. The greedy method first chooses the 89 to add to the set of angles and afterwards 60. The set of equidistantly distributed angles reaches lower cost function value (given in the captions).

6.1.2 Reconstruction with 10 projection angles

The results for the phantoms with 10 projection angles, are shown in Figure 17. It can be seen that the rectangle reconstructions improve less than the other two phantoms, due to the fact that the rectangle can be well reconstructed from only two angles. For the other two phantoms, there is more information to be gained by extra angles.

The coordinate descent method on the result of the greedy algorithm performs best on all phantoms with 10 projection angles. Note though that this method is slow, as in each iteration all angles are updated and many evaluations of the cost function L are necessary.

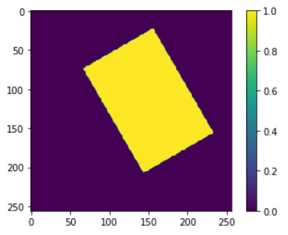
For the ten angles, as for the two angles, the ensemble Kalman method performs the worst of our three algorithms. Note that the result of the ensemble Kalman method depends on the initial random particles. Therefore it would be good to try different random seeds, as each will give a different result. Another improvement would be to increase the number of particles. Moreover, the ensemble Kalman method has more parameters (i.e. step size) that may improve the angle set and should be investigated further. The ensemble Kalman method will be useful on images that have a very irregular cost function, i.e. with multiple local minima, as it is flexible and the iterations are fast.

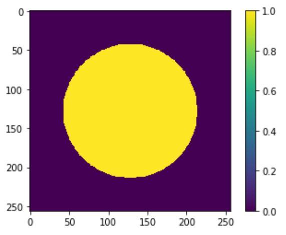
6.1.3 Comparison with equidistant angle selection

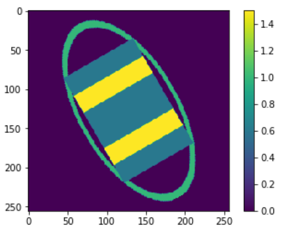
To investigate whether the proposed angle selection methods indeed give lower cost function evaluation than equidistantly distributed angles, we performed all methods on all phantoms for a range of total angle numbers. The results are shown in Figure 18. For the calculations, again 5 iterations of SIRT have been used. For the ensemble Kalman method the number of particles is equal to ten times the number of angles. The initial angles for the coordinate descent method are the angles found by the greedy algorithm.

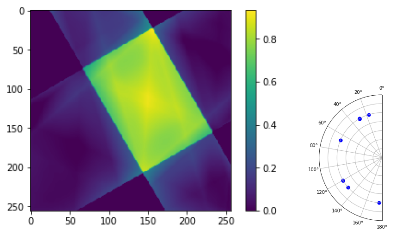
Examining Figure 18(a), it can be seen that especially the greedy and coordinate method have lower cost function value than the equidistant method. The peak for three angles is most likely due to the fact that the rectangle can be well reconstructed from only two angles. As the number of SIRT iterations is low, the third angle does not improve the reconstruction.
For the circle phantom, we have previously discussed that equidistant angles give the best reconstruction. Therefore, we did not expect the angle sets found by our algorithms to perform better than using an equidistant angle set. From Figure 18(b), it is clear however, that the coordinate descent method performs just as well as the equidistant angle selection. As the coordinate descent method does reach lower cost evaluation on the other phantoms, we can conclude that it does not perform worse than the equidistant angles for our test cases. This is a promising result. The coordinate descent method can be used on any set of initial angles, including the equidistant set. This will never increase the cost function evaluation.

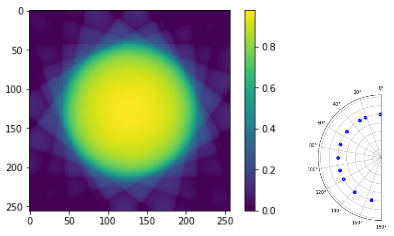
The results for the boat phantom also show that, for this more complicated phantom, the greedy and coordinate descent method yield a lower cost function value than the equidistant angle selection. The peak at three angles can again be explained in the same way as for the rectangle, as the boat phantom has rectangular features that resemble the rectangle phantom. We would expect the improvement of selecting the angles in a non-equidistant way, to become less when the number of angles increases. For both the rectangle and boat phantom it can be seen however, that even for 20 angles, the coordinate method reaches lower cost function evaluation than equidistant angle selection.

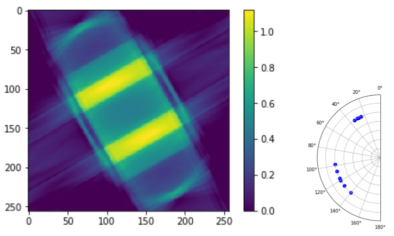

(a) True

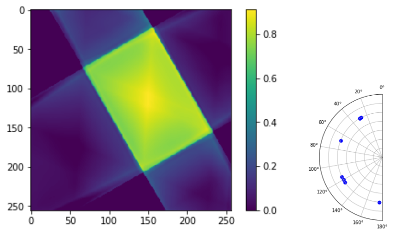

(b) True

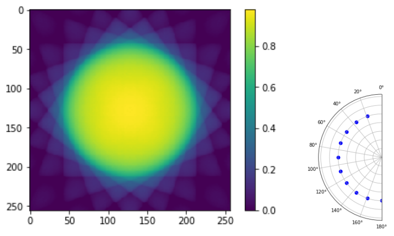

(c) True

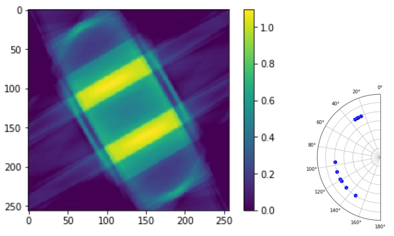

(d) Greedy, L = 17.98

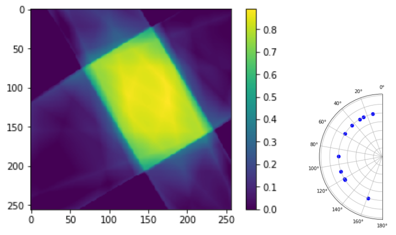

(e) Greedy, L = 22.24

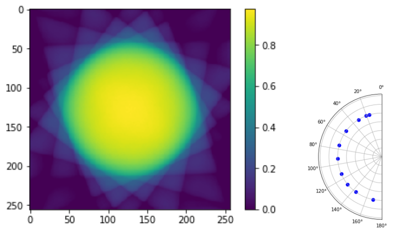

(f) Greedy, L = 30.05


(g) Coordinate, L = 17.66


(h) Coordinate, L = 21.95


(i) Coordinate, L = 29.83


(j) Ensemble, L = 19.46


(k) Ensemble, L = 22.48

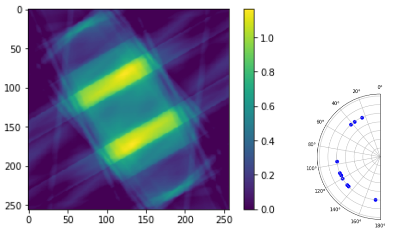
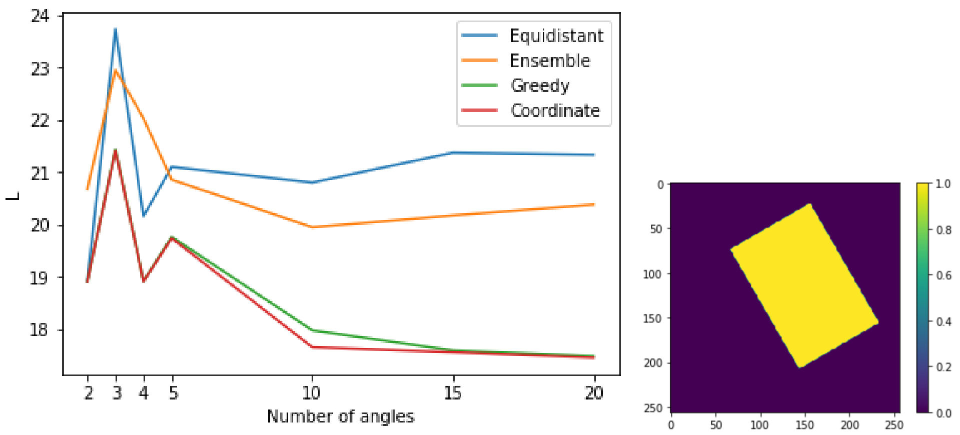
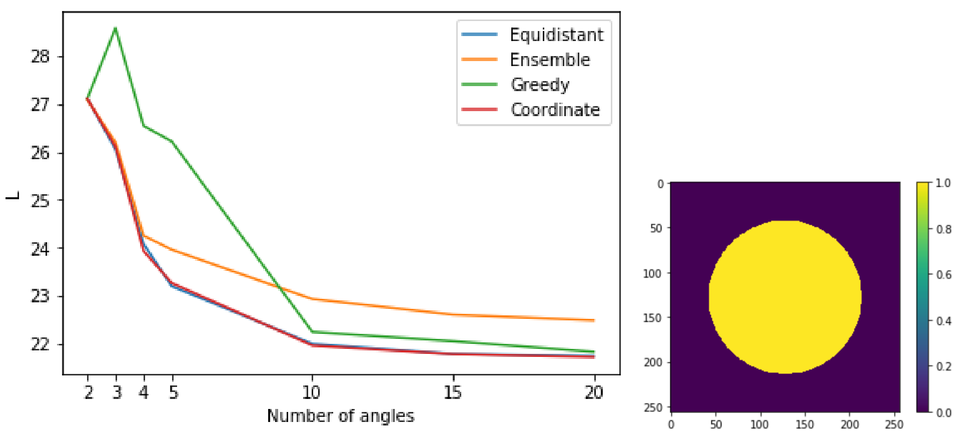

(l) Ensemble, L = 31.18

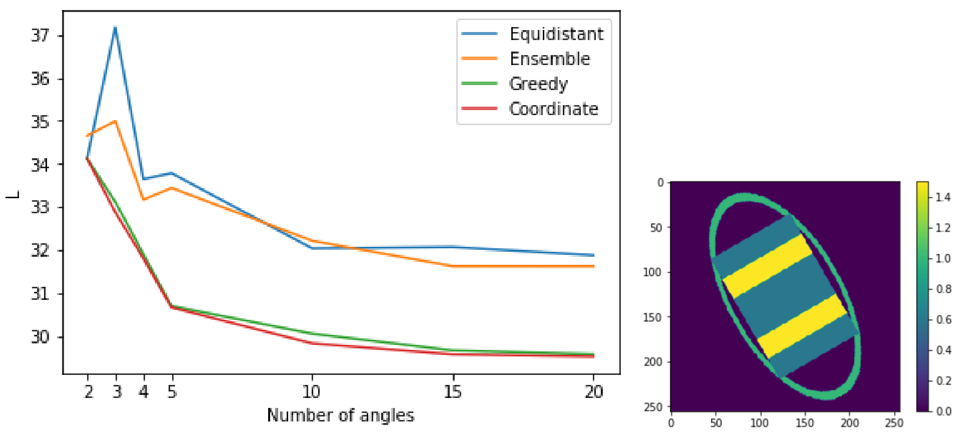
Figure 17: Results of the angle selection methods for ten angles. Top: true images, below: reconstructions using the angles found by the methods, chosen angles shown to the right of the reconstruction, corresponding cost function value L given in the captions.



(a) Rectangle



(b) Circle



(c) Boat

Figure 18: Performance of the angle selection methods and equidistant angles. On the y-axis is the resulting cost function value for choosing the angles with either one of the three proposed methods or equidistantly. The number of angles chosen for evaluations are shown on the x-axis. 5 iterations of SIRT within reconstruction function G. Evaluated angles: $\{2,3,4,5,10,15,20\}$.

6.2 Training set simulations

To test the theory in Section 3.2, a true image and training samples from the same probability distribution were generated (see Section 5.2.1). Figure 19 shows the true image.
The training samples are randomly generated, so a random seed is set to ensure the same results for replication. The generated diamond shapes all have the same center and differ in length, width and intensity. The total intensity of each diamond is fixed and the same for all diamonds. In Figure 20, we show the cost functions for the true image, an average over 10 training samples with variation in both horizontal and vertical direction and an average with variation only in the vertical direction. Each cost function is scaled by subtracting its minimum and by dividing by the maximum of the true cost function. Contour lines are drawn at each decimal value.
The last cost function simulates that the vertical projection is already taken into account and therefore the width of the diamond is known. In that case the only uncertain factor is the height of the diamond. For each of the cost functions the value is shown of collecting data with angle set $\boldsymbol{\theta} = \{0, \theta_1, \theta_2\}$, with θ_1 on the x and θ_2 on the y axis.

In Figure 20, the true cost function, the cost function for 10 training samples with variation in both directions and the cost function with variation only in the vertical direction, are shown. The variation in the horizontal direction is higher (30%) than the variation in vertical direction (10%). The reconstructions using the angles corresponding to the minimum of each cost function are shown below that cost function. The results show that using training samples the angles can be more accurately chosen than without training samples, in which case equidistant angles would be chosen.

The angles that define the minimum in Figure 20(b) give a reconstruction (Figure 20(e)) with lower cost function value than a reconstruction with equidistantl angles (Figure 20(g)) as the prior knowledge about the training samples is taken into account. From Figure 20(c), we can conclude, that once one angle is scanned and this prior knowledge can be taken into account before scanning the next angle, this gives great opportunities to improve the surrogate cost function. The minimum of the average is closer to the real minimum. It can also be seen that the shape of the cost function Figure 20(c) resembles that in 20(a) more closely than 20(b) and that the cost function evaluation is lower. The cost functions illustrate the use of taking one projection to incorporate the information of that projection into the training samples.

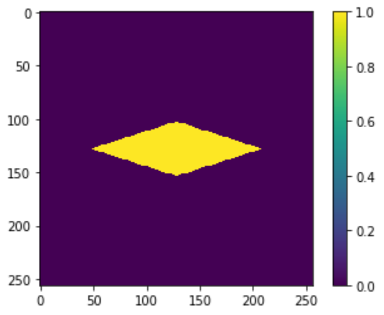
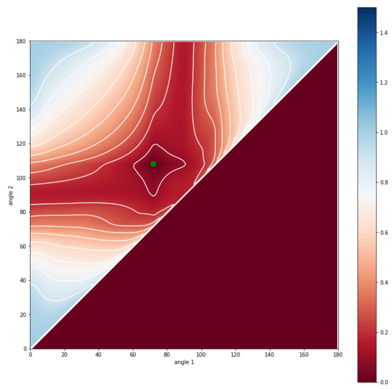
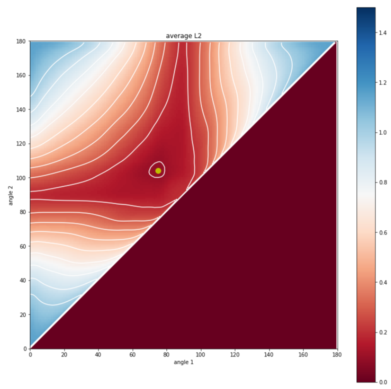


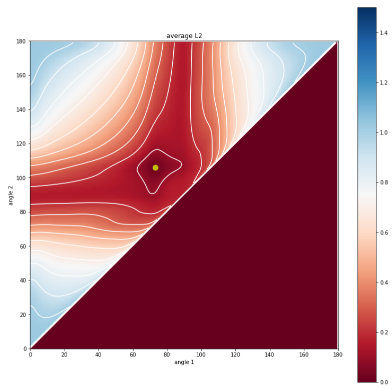
Figure 19: True image for training sample simulations.



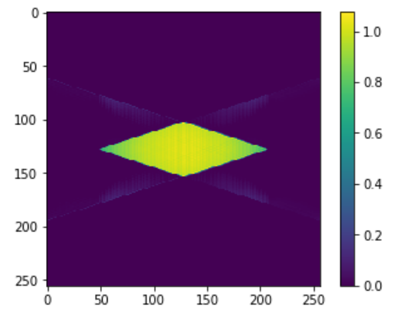
(a) True cost function. Minimum = {72,108}



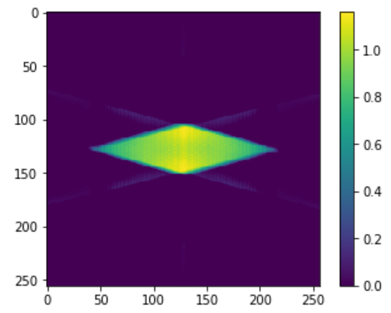
(b) Average, 30% variation in horizontal direction, 10% in vertical direction. Minimum = {75,104}



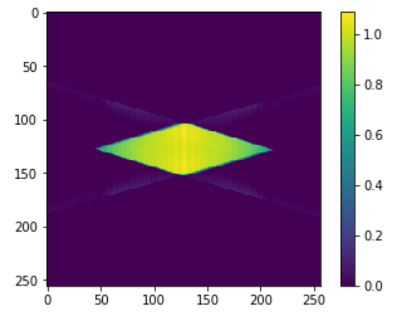
(c) Average, 30% variation in vertical direction only. Minimum = {73,106}



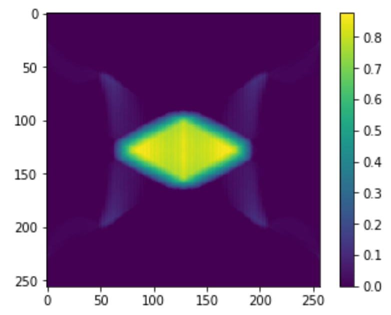
(d) Reconstruction based on true cost function minimum.
$L(\{0, 72, 108\}) = 3.76$



(e) Reconstruction based of average cost function minimum.
$L(\{0, 75, 104\}) = 7.41$



(f) Reconstruction based on average cost function minimum.
$L(\{0, 73, 106\}) = 5.11$



(g) Reconstruction based on an equidistant angle set.
$L(\{0, 60, 120\}) = 15.22$

Figure 20: (a) Cost function for the true image. (b) Cost function average over 10 training samples from the same distribution with 30% variation in horizontal direction, 10% in vertical direction. (c) Cost function average over 10 training samples, variation 10% in vertical direction only. Cost function images have 10 contour lines and use 100 iterations of SIRT within reconstruction function G. (d)-(f) Reconstructions using the angles corresponding to the minimum of the cost functions. (g) Reconstruction using equidistant angles.

6.3 Real data example

Ideally we would like to test the angle selection algorithms and the training sample simulations on real data. As a first step towards this goal, we will use data from a scan using the FleX-Ray scanner at Centrum Wiskunde & Informatica (CWI) to create a phantom that resembles a real object. The phantom is a 'golden standard recontruction', created by reconstructing the high quality data from the scanner with 600 iterations of the SIRT algorithm (Algorithm 4).
Part of this section and the corresponding programming in Python, are based on a research project for the Mastermath course Inverse Problems in Imaging, given between February and June 2018 by Tristan van Leeuwen (University of Utrecht) and Christoph Brune (University of Twente). The subject of the project was the reconstruction of a tree stem in horizontal direction, i.e. such that the full year rings were visible, and exploiting the radial symmetry of the year rings. The project was carried out in collaboration with Valesca Peereboom and Fredrik Gustafsson. Data taken for that project also included the data presented here, of the tree stem in vertical orientation. This data was not used for the project however.

In this section we will first discuss the background of scanning wood to see year rings, then explain our sample and data and describe the necessary pre-processing steps of the data. Next, we discuss how the golden standard reconstruction was made and we show the results of the angle selection methods compared to the equidistant angle selection.

6.3.1 Background

The density of wood is an interesting characteristic, especially in industrial purposes, as it indicates the quality of the wood. A way to obtain this knowledge of a wooden object is by a X-ray scanner. Such a scanner determines the density by the measured intensity transmitted trough the object. X-ray scanning can also be used for internal defect detection [10]. One practical example is from modern sawmills, where logs are scanned and reconstructed in 3D, and a high yield cut is calculated [18].

If it is possible to clearly see the year rings in a CT reconstruction, this can be be used to calculate the age of the wood (for example of art objects). CT offers a non-destructive method for dating wood, whereas in more conventional dendrochronology a piece of the object has to be cut out to be able to see the rings. Thus, the application of CT to dendrochronology will allow more objects to be dated. To see the year rings, the image resolution and intensity of the X-ray beam are important factors [5].

The year rings are also interesting to climatologists, as via these the growth of trees can be used for environmental impact assessment and climate back-forecasting. Within one year, early wood is less dense than late wood and thus the year rings are clearly visible in scans of wooden logs. Uncertainty in the slices is highest where there are irregularities (e.g. a branch) [9].

6.3.2 Sample and data

The primal goal is to reconstruct a two dimensional internal slice of a tree stem. For that, we will use data from the FleX-Ray scanner to reconstruct a slice of the object shown in Figure 21. The object is a 5.5x4.5x3cm almost cylindrical wooden block.
The FleX-Ray scanner uses an X-ray source and a detector that registers the intensity transmitted through the object. The emitted beam type is a cone beam. The object rotates on a platform between source and detector. For our data, 1800 equally spaced projections were taken. The data consists of two dimensional images with a high intensity at places with soft material and low intensity at places with harder material. The three dimensional internal structure of the object can be reconstructed from these images. For our simulations we will however use a 2D slice through the tree stem.
In the next section we will describe the data obtained by the FleX-Ray scanner and how this data should be pre-processed before reconstruction.

Figure 21: The scanned tree stem.

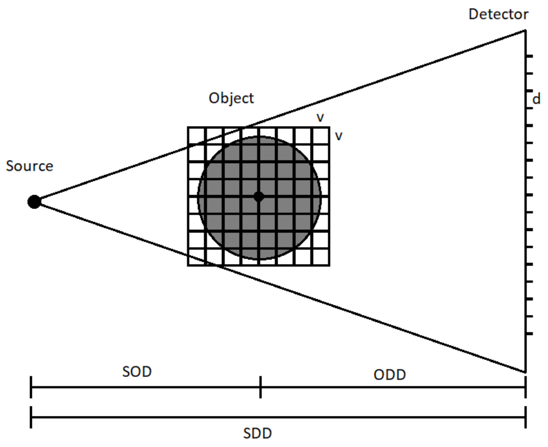


Figure 22: Schematic representation of the setup.

6.3.3 Pre-processing

As previously discussed in Section 3, the intensity of the X-ray beam through an object with attenuation function $\mu(x)$ follows the Lambert-Beer Law:

$$I = I_0 e^{-\int_L \mu(x)dx}, \tag{16}$$

where I is the measured intensity at the detector, I_0 the intensity of the beam as it leaves the X-ray source and L the length the ray travels through the object. The data from the FleX-Ray scanner needs to be pre-processed to correct for the dark- and flatfield and the fact that the center of the object may not coincide exactly with the center of rotation. Thirdly, a correction of magnification factors is needed before the data can be used within the Astra Toolbox.

- **Dark- and flatfield correction**
  The input of the data for reconstruction with the Astra Toolbox needs to be $\int_L \mu(x)dx$, the integral over the attenuation function. Moreover, the data needs to be corrected for the *darkfield* D_0, the nonzero data the detector always measures even if the source is off. This can for example be due to background radiation or electrical current. The darkfield is measured once while the source is off. As this bias occurs on every measurement, it needs to be subtracted from each measurement. Moreover a *flatfield* measurement is taken before and after the data collection. This is the measurement of I_0, in other words the intensity of the beam without any object in its way. We correct for these fields and process the data to follow the rewritten version of Equation 16:
  $$d = \int_L \mu(x)dx = \log(\frac{I_0 - D_0}{I - D_0}). \tag{17}$$

- **Center of rotation correction** As the center of the object in the scanner will not exactly match the center of rotation, the data needs to be realigned, so that the center of the object is in the center of rotation. The data is represented as a sinogram, a matrix in which each horizontal line corresponds to one measurement. We calculate the sum over each row (the sum over each detector pixel) and then find the center of mass. This center of mass of the sinogram is then shifted using an Astra function to shift all the measurements so that the center of mass coincides with the center of rotation.

- **Magnification correction** The metadata from the scanner give the Source-to-Detector-Distance (SDD), the Source-to-Object-Distance (SOD) and the detector pixel height and width d. See Figure 22 For the Astra input we need the Object-to-Detector-Distance (ODD), $ODD = SDD - SOD$. In Astra moreover, the pixel size of the discretization of the object is automatically set to 1. Therefore, the other values have to be scaled to match. The magnification factor is $\frac{d}{v} = \frac{SDD}{SOD}$. From this formula we calculate the 'actual' v and scale everything accordingly:
  $$d' = \frac{d}{v} \qquad SDD' = \frac{SDD}{v} \qquad SOD' = \frac{SOD}{v} \qquad ODD' = \frac{ODD}{v}.$$
  The rescaled values are used as input for Astra reconstruction algorithms.

6.3.4 Golden standard reconstruction

As our algorithms have been designed for two-dimensional phantoms, we take the slice exactly in the middle of the scan. This ensures that the beam can be treated as a fan beam in 2D. The slice then lies in the same plane as the center of the beam, which means that the incoming rays are not tilted. The golden standard reconstruction, that will be used as phantom, is shown in Figure 23. In order to make this reconstruction, we used the Astra Toolbox, with fan beam projection geometry. The reconstruction is made with 600 iterations of SIRT. This image is in the following paragraphs used to test our algorithms for lifelike wood data.

6.3.5 Angle selection

The wood phantom, that was created from the real data, clearly contains main directions because of the visible year rings. In Figures 24 and 25, the reconstructions of the wood phantom are shown for two and ten angles respectively. The angles are chosen by the greedy, coordinate descent and ensemble Kalman algorithms and equidistantly. Additionally, the difference $|\hat{\mathbf{f}} - \mathbf{f}|$ is shown. For the images, the same simulation approach as for the other phantoms has been used. This means that from the wood phantom data was generated by using the forward projection of the Astra Toolbox, not the real data from the scanner. The reason for this, is that the algorithms have been designed to take in a true image and simulate the data. Moreover, the algorithms assume parallel beam geometry. The data given by the scanner comes from a fan beam. This changes not only the geometry, but also the range in which we search for the best angle. For parallel beam, we have restricted the search to angles between 0 and 180 degrees. For fan beam this would not be a good strategy, all angles should be taken into account. Because of time limitations, we have chosen to investigate a true image, created from real data. We will leave the use of real data for future research. Within the angle selection algorithms, we simulate parallel beam data from the golden standard reconstruction.

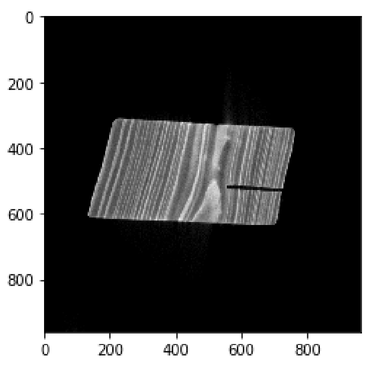
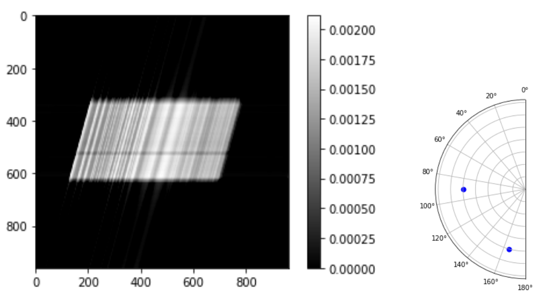
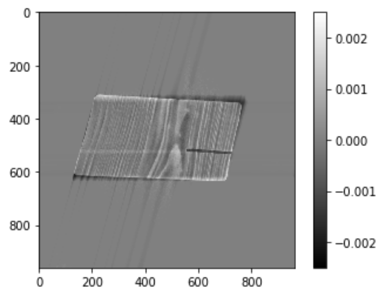
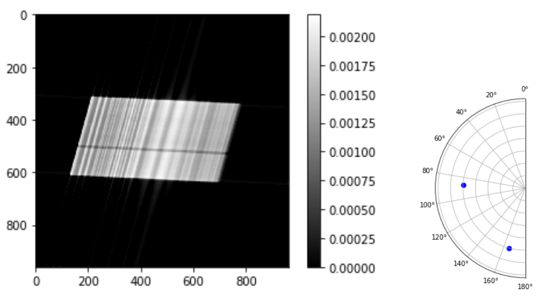


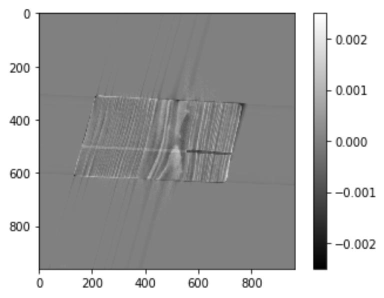
Figure 23: The golden standard reconstruction, using 600 iterations of the SIRT algorithm.

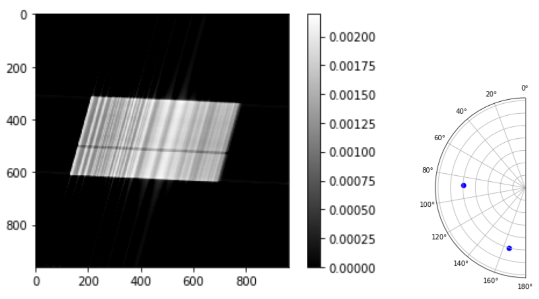
In the figures it can be seen that even for this more complicated phantom, two angles can give a reconstruction in which the internal structure is visible. The lines corresponding to the year rings are not sufficiently clear to count however. In Figure 25, it is interesting to see that the angles that are chosen are close and the cost function is lower than the equidistant angle selection. As the number of angles increases and the reconstruction becomes better, the image that shows the difference between the true image and the reconstruction, should become less structured and eventually look like random noise. We can indeed see that for the 10 angle reconstructions the difference contains less structure than that of the reconstructions with 2 angles.
These reconstructions show that few projection angles can suffice to be able to count the wood rings in a tree stem. This is an interesting result for dendrochronology. Valuable and fragile objects (such as art objects) can be subjected to less radiation, if the angles are chosen in the right way. We see that a few projections suffice to give information on the direction of the year rings. In wood the lines are often approximately aligned, which makes it an especially interesting material to investigate for angle selection. Different wooden logs or stems have the same features, the year rings have the same structure. This gives opportunities to create training data and to use prior knowledge within the selection of projection angles.

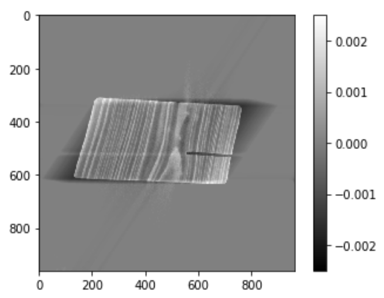

(a) Greedy, $L = 0.091$

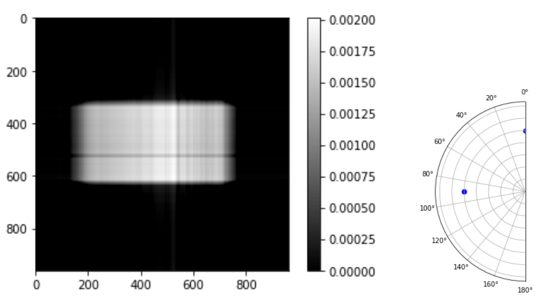

(b) Difference between reconstruction and true image


(c) Coordinate, $L = 0.074$


(d) Difference between reconstruction and true image


(e) Ensemble, $L = 0.074$


(f) Difference between reconstruction and true image


(g) Equidistant, $L = 0.11$

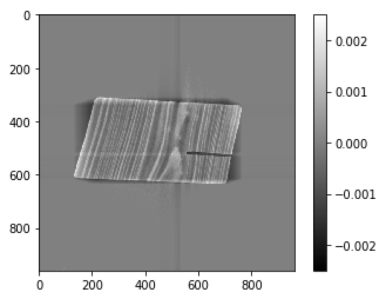
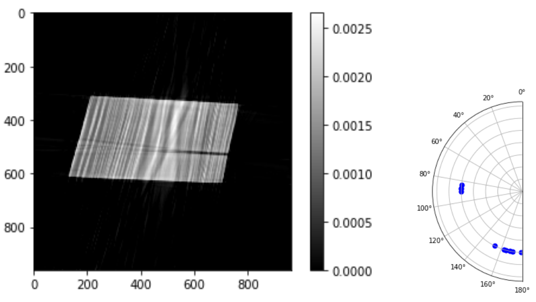
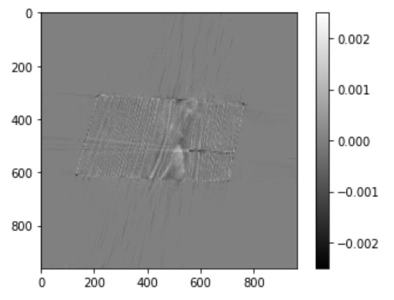

(h) Difference between reconstruction and true image

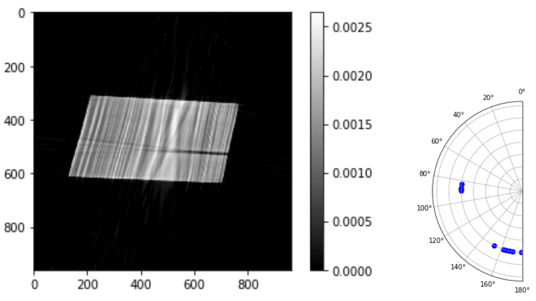
Figure 24: Reconstructions using two angles, found by the angle selection algorithms and a reconstruction using equidistant angles. Next to that, the difference between the reconstruction and the golden standard reconstruction is shown.



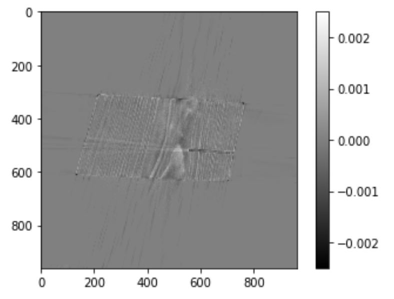
(a) Greedy, $L = 0.047$



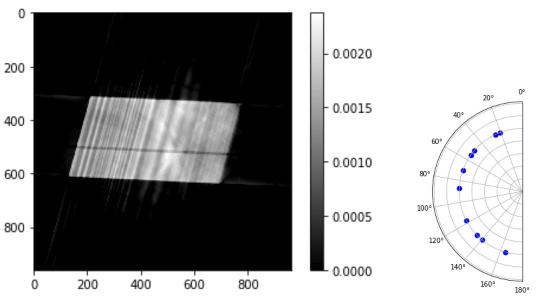
(b) Difference between reconstruction and true image



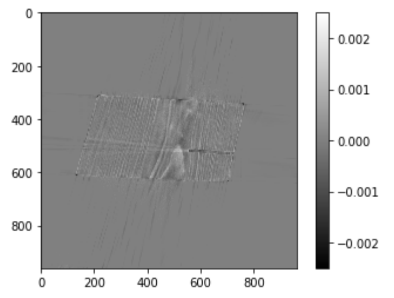
(c) Coordinate, $L = 0.047$



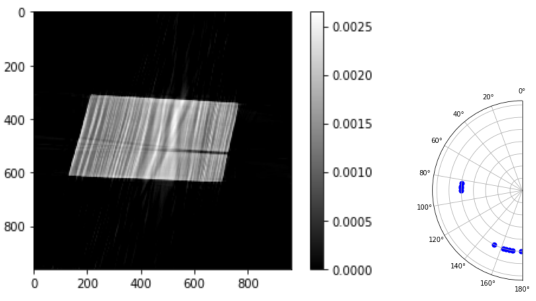
(d) Difference between reconstruction and true image



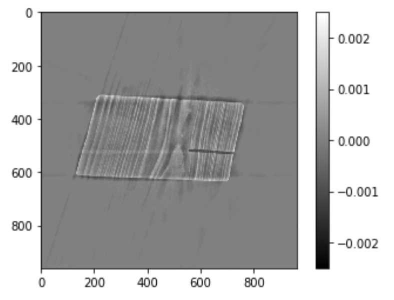
(e) Ensemble, $L = 0.071$



(f) Difference between reconstruction and true image



(g) Equidistant, $L = 0.088$



(h) Difference between reconstruction and true image

Figure 25: Reconstructions using ten angles, found by the angle selection algorithms and a reconstruction using equidistant angles. Next to that, the difference between the reconstruction and the golden standard reconstruction is shown.

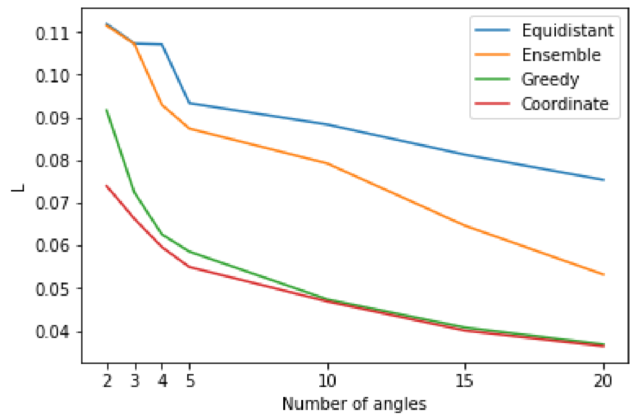


Figure 26: Performance of the angle selection methods and equidistant angles. On the y-axis is the resulting cost function value for choosing the angles with either one of the three proposed methods or equidistantly. The number of angles chosen for evaluations are shown on the x-axis. 100 iterations of SIRT within reconstruction function G. Evaluated angles: $\{2,3,4,5,10,15,20\}$.

6.3.6 Comparison with equidistant angle selection

In Figure 26 the results of the cost function evaluation for all methods and for different total number of angles, can be seen. The methods are the same as in Section 6.1.3, with the exception that for these results 100 iterations of SIRT have been used.
These results show that, for the wood phantom, all methods perform better than the equidistant angle selection. We expected non-equidistant projection angles to perform better than the equidistant projection angles, as the phantom has main directions due to the year rings. The alternative angle selections give lower cost function evaluation, also for higher total number of angles. The year rings are visible in the reconstructions with a low number of angles. This is a promising result for dendrochronology, as we have shown that only a few projections can suffice to be able to distinguish the year rings to determine the age of the wood.

7 Conclusion and Discussion

In this thesis, we have discussed the optimisation of projection angle selection in Computed Tomography. Given a maximum number of projections, the aim was to identify the projection angles that give maximal information gain, when added to the existing set of angles. In Section 4 we have proposed three approaches: a greedy algorithm, a coordinate descent algorithm and an adaptation of the ensemble Kalman Filter method.

The algorithms were tested on three different computer generated phantoms and one phantom generated from data from the FleX-Ray scanner. Which algorithm is best suited to tackle the angle selection problem, may be object dependent. The greedy method followed by the coordinate descent method gives satisfactory results, especially on our simple phantoms. The coordinate descent method gives the lowest cost function evaluation of the three algorithms, but is the most computationally heavy of the three as it needs many evaluations of the cost function.
The ensemble Kalman method is a fast algorithm that may be very useful if the cost function contains several local minima. To improve this algorithm, it is possible to also perform a coordinate descent on the resulting set of angles. The ensemble Kalman method contains many parameters that need to be adjusted to each object. More investigation into the choice of parameters and the connection with the object, is required to understand how to use this algorithm most efficiently. If the settings of the parameters are better understood, the ensemble Kalman method promises to be a useful and fast algorithm.

Next, we showed results on reconstructions of the wood phantom in Section 6.3, that was based on real data from the FleX-Ray scanner at CWI. These results show that for data containing main features, such as year rings in wood, the algorithms find the angles that correspond to these main directions. The reconstructions of the data using these optimised projection angles fit the original phantom better than reconstructions using the same number of equidistant angles.

The reconstructions based on the angle sets chosen by the algorithms, serve to show that it is possible to choose angles in a 'smart' way. We have shown that for some phantoms it is possible to unravel the internal structure of object with few angles. Moreover we have shown that the conventional way of choosing the projection angles equally spaced, is not optimal when facing limited angle reconstructions. The greedy and coordinate descent methods have been shown to give lower cost function evaluation than the regularly used equidistant approach. This has been investigated for low numbers of total angles. We expect the added value of performing one of the algorithms to choose the projection angles to become less, when the number of angles increases. Incorporating prior knowledge on the subject, such as the alignment of the object in the scanner, can be of great use to improve the selection.

The results on the cost function evaluation of the training data, show that if training data is available, this can be of great help to incorporate prior knowledge of the shape of the object in the optimisation process. As in real life applications the true image is not available, this approach gives a good starting point for further investigation. We have shown, that the effect of taking one projection and taking the gained information into account, can improve the training set. If this can be implemented in a fast way, it may be possible to calculate the next angle to be chosen real-time, adjusting the cost function approximation after each projection.

The approach for projection angle selection in this thesis is relatively new, therefore there are many interesting possibilities for continuing research on this subject. Below we list some ideas.

- Investigation into the optimal settings of the parameters of the ensemble Kalman method.
- Research into the performance of the algorithms for images with more whimsical, e.g. containing more local minima, cost function evaluations.
- How to design or create training data. One possibility might be to take high quality scans of similar objects and treat the reconstruction of those as true training images.
- Extension of the algorithms work on real data and 3D objects. The object rotates only around its own axis. The relative vertical position of source, object and detector stays the same. The set of possible projection angles does therefore not change with the extension. The optimisation problem would not

change significantly, only the reconstructions and cost functions would become more complicated. The Astra Toolbox supports the necessary 3D reconstructions.

- Dynamical scanning approaches: using the information of each projection to decide which projection angle should be chosen next. Eventually the goal would be to compute the next projection angle in real time.
- Add angles iteratively untill a certain precision is reached, instead of a total number of angles fixed beforehand.
- In Appendix A we have investigated the influence of noise in the data on the cost function. When noise is present, the cost function becomes less smooth. This makes it more difficult to use the proposed angle selection algorithms, as with these a local minimum instead of a global minimum may be found. The proposed noise generation method is artificial and therefore it is necessary to investigate the influence of noise in real data on the cost function and angle selection methods more fully.

In conclusion, the selection of projection angles in a different way than equidistant, is a promising approach to increase the quality of reconstructions with few projection angles. There are many interesting possibilities for further investigation of the subject.

List of symbols and abbreviations

$\mathbf{A}$	Forward operator of the data model 4
$\mathbf{d}$	Data (see data model 4)
$\mathbf{E}$	The ensemble containing particles (angle sets) of the Ensemble Kalman Filter (Algorithm 3)
$\mathbf{f}$	True image (represented as a vector of size n^2 if the visual representation is a matrix of size $m = n \times n$)
f_i	One pixel from the true image
$\hat{\mathbf{f}}$	The reconstruction of data acquired using model 4, the same shape as $\mathbf{f}$
$\hat{f}_i$	One pixel from the reconstruction
$G(\theta)$	Function that gives the reconstruction $\hat{\mathbf{f}}$ based on the projections $\boldsymbol{\theta}$
I_0	Intensity of the X-ray beam on leaving the source
I	Intensity of the X-ray beam after passing through a material
$L(\theta)$	The cost fuction that gives the L^2-norm of a reconstruction $\hat{\mathbf{f}}$, see Equation 5
μ	Attenuation coefficient of the material
$\boldsymbol{\theta}$	A set of N_θ projection angles: $\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots, \theta_{N_\theta}\}$
N_θ	The total number of projection angles, the size of set $\boldsymbol{\theta}$
$N_{detector}$	The total number of detector pixels
N_{train}	The total number of training samples
n_E	The total number of particles (angle sets) in the Ensemble Kalman method (Algorithm 3)
$\mathbf{p}_i$	One particle in the ensemble of the Ensemble Kalman method

References

[1] W. van Aarle, W. J. Palenstijn, J. Cant, E. Janssens, F. Bleichrodt, A. Dabravolski, J. De Beenhouwer, K. J. Batenburg, and J. Sijbers, *Fast and Flexible X-ray Tomography Using the ASTRA Toolbox*, Optics Express, 24(22), pp. 25129-25147, 2016

[2] W. van Aarle, W. J. Palenstijn, J. De Beenhouwer, T. Altantzis, S. Bals, K. J. Batenburg, and J. Sijbers, *The ASTRA Toolbox: A platform for advanced algorithm development in electron tomography*, Ultramicroscopy, 157, pp. 35-47, 2015

[3] W. Van Aarle, *Tomographic segmentation and discrete tomography for quantitative analysis of transmission tomography data*, PhD Thesis, University of Antwerp, 2012

[4] K.J. Batenburg, W.J. Palenstijn, P. Baláz, J. Sijbers, *Dynamic angle selection in binary tomography*, Comput. Vis. Image Understand., no. 117, pp. 306-318, 2013

[5] J. Bill, A. Daly, Ø. Johnsen, K. Dalen, *DendroCT - Dendrochronology without damage*, Dendrochronologia, 30, pp. 223-230, 2012

[6] T.M. Buzug, *Computed Tomography: from photon statistics to modern cone-beam CT*, Springer-Verlag Berlin Heidelberg, 2008

[7] A. Chambolle, T. Pock, *An introduction to continuous optimization for imaging* Acta Numerica, no. 25, pp.161-319, 2016

[8] G. Chen, J. Tang, S. Leng, *Prior image constrained compressed sensing (PICCS): A method to accurately reconstruct dynamic CT images from highly undersampled projection data sets*, Med. Phys. 35(2), pp. 660-663, 2008

[9] Dutilleul, P., Wen Han, L., Beaulieu, J., How do trees grow? Response from the graphical and quantitative analyses of computed tomography scanning data collected on stem sections, C.R. Biologies, Elsevier Masson SAS, 337, pp.391-298, 2014

[10] C. Freyburger, F. Longuetaud, F. Mothe, T. Constant, J. Leban, *Measuring wood density by means of X-ray computer tomography*, Annals of Forest Science, 66(8), 2009

[11] E.F. Garman, M. Weik, *X-ray radiation damage to biological macromolecules: further insights*, Journal of Synchroton Radiation, no. 24, pp. 1-6, 2017

[12] J. Hsieh, *Computed Tomography: principles, design, artifacts, and recent advances*, SPIE PRESS, 2003

[13] P. Ghosh, *Scan technique reveals secret writing in mummy cases*, http://www.bbc.com/news/science-environment-42357259, accessed 09-04-2018

[14] J. Gregor and T. Benson, *Computational analysis and improvement of SIRT*, IEEE Transactions on Medical Imaging, vol. 27, no. 7, pp. 918-924, 2008

[15] M.A. Iglesias, K.J.H. Law and A.M. Stuart, *Ensemble Kalman methods for inverse problems*, Inverse Problems 29, 045001, 2013

[16] G. Van Kaick, S. Delorme, *Computed tomography in various fields outside medicine*, Eur. Radiol. Suppl., 15[4], D74-D81, 2005

[17] P.A. Midgley, R. E. Dunin-Borkowski, *Electron tomography and holography in materials science*, Nature Materials, Vol. 8, 271-280, 2009

[18] Microtec CT Log 360 X-ray CT-Sawing Optimization `https://www.youtube.com/watch?v=xK4CdNT3DK4`

[19] National Cancer Institute (USA), *Computed Tomography (CT) Scans and Cancer*, https://www.cancer.gov/about-cancer/diagnosis-staging/ct-scans-fact-sheet, accessed 09-04-2018

[20] L. Ruthotto, J. Chung and M. Chung, *Optimal experimental design for constrained inverse problems*, ArXiv, 2017

[21] K.B. Petersen, M.S. Pedersen, *The matrix cookbook*, https://www.math.uwaterloo.ca/ hwolkowi/matrix-cookbook.pdf, version: november 15, 2012

[22] C. Schillings, A. Stuart, *Analyisis of the ensemble kalman filter for inverse problems*, SIAM J. Numer. Anal., Vol. 55, no. 3, pp. 1264-1290, 2016

[23] R. Smith-Bindman et al. *Ratiation dose associated with common computed tomography examinations and the associated lifetime attributable risk of cancer*, Arch Intern Med., December 14, 169(22): 2078-2086, 2009

[24] E.Y. Sidky, C. Kao, X. Pan, *Accurate image reconstruction from few-views and limited-angle data in divergent-beam CT*, ArXiv, date: April 28, 2009

[25] The British Museum, *Scanning Sobek*, http://www.britishmuseum.org/whats_on/exhibitions/scanning_sobek.aspx#scans, accessed 09-04-2018

[26] The Field Museum, *Computed Tomography (CT) Scanning*, https://www.fieldmuseum.org/science/research/area/conserving-collections/examination-documentation/computed-tomography-ct, accessed 09-04-2018

[27] University College Londen, university website, *Deep Imaging Egyptian Mummy Cases*, http://www.ucl.ac.uk/dh/projects/deepimaging/, accessed 09-04-2018

Acknowledgements

I would like to thank my supervisors Floske Spieksma, Joost Batenburg, Felix Lucka and Alexander Kostenko for their support during my project and their useful feedback. Thank you, Joost Batenburg, for giving me the opportunity to experience working at CWI, which turned out to suit me very well. I look forward to our future collaboration. Thank you, Willem Jan Palenstijn and Maureen van Eijnatten, for all the fruitful discussions, the fun at work and moreover, for helping me to shape my ideas about my future career. I would also like to thank my other colleagues from the Computational Imaging Group at CWI for making me feel so welcome and the atmosphere at work so enjoyable. And thanks to Ionica Smeets, for your support, the interesting and fun collaborations during my studies in Science Communication and for directing my attention to the research group of Joost Batenburg for a thesis internship.

I would like to thank my parents, Arda van der Veer and Wim Bossema, for encouraging me, motivating me for my studies as well as my hobbies and always being proud of me. Last but not least, thank you, Frank Bieger, for supporting me in everything I undertake and encouraging me to be the best possible version of myself.

Appendices

A Additional research: Noise simulation

To be able to simulate data more realistically, it is common to add noise to the measurement data. This is done by adding additive Gaussian (random) noise to the sinogram after applying the forward operator **A**. This is the simplest way of adding independent noise. In reality the noise is not always independent, it may for example be the case that detector pixels that are close influence each other. In this section our model for noisy data will be described, starting with the theory behind continuous noise and showing how this noise influences the cost function L afterwards.

A.1 Noise model

The additive noise for each pixel i is of the form

$$N_i(\theta) = \sum_{k=1}^{K} (v_{ik} \cos(\frac{2\pi}{180} k\theta) + w_{ik} \sin(\frac{2\pi}{180} k\theta)),$$

where v_{ik} and w_{ik} are normally distributed random variables. This representation of the noise is chosen, because the cosine and sine of $2\pi/180 * n$, $n \in 1, 2, \dots$ form a basis for the $L^2[0, 180]$ space and thus we can extend the evaluation of picking random noise on a grid to a continuous function by using a sum of sines and cosines. We thus create a continuous noise function that generates noise for every possible angle θ and that is the same each time we use the same angle. The v_{ik} and w_{ik} are normally distributed with mean 0 and standard deviation $\frac{\alpha P}{\sqrt{K}}$, where P is the peak of the data, i.e. the highest value in the data resulting from the forward projection. The noise for each angle is then distributed with mean 0 and standard deviation equal to a fixed percentage of P. This can be shown by using the rules for adding normally distributed random variables and multiplying by scalars. Firstly:

$$v_{ik} \sim \mathcal{N}(0, (\frac{\alpha P}{\sqrt{K}})^2)$$

$$v_{ik} \cos(\frac{2\pi}{180} k\theta) \sim \mathcal{N}(0, (\frac{\alpha P}{\sqrt{K}})^2 \cos^2(\frac{2\pi}{180} k\theta)).$$

The same applies to the sine part of the sum:

$$w_{ik} \sin(\frac{2\pi}{180} k\theta) \sim \mathcal{N}(0, (\frac{\alpha P}{\sqrt{K}})^2 \sin^2(\frac{2\pi}{180} k\theta)).$$

Combining the last two distributions gives:

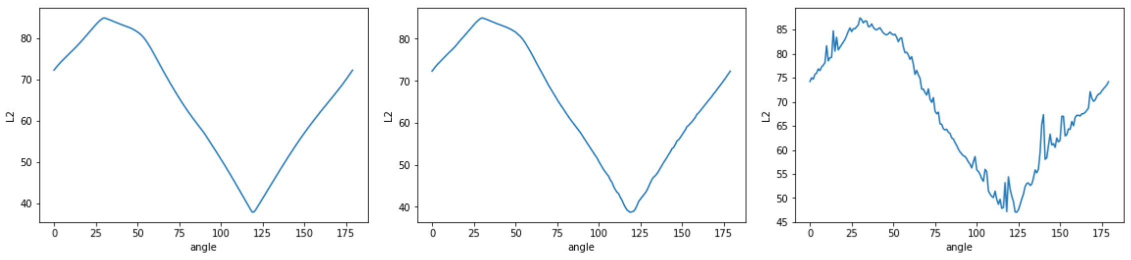
$$v_{ik} \cos(\frac{2\pi}{180} k\theta) + w_{ik} \sin(\frac{2\pi}{180} k\theta) \sim \mathcal{N}(0, (\frac{\alpha P}{\sqrt{K}})^2 (\sin^2(\frac{2\pi}{180} k\theta) + \cos^2(\frac{2\pi}{180} k\theta)))$$

$$\sim \mathcal{N}(0, (\frac{\alpha P}{\sqrt{K}})^2).$$

Furthermore:

$$\sum_{k=1}^{K} (v_{ik} \cos(\frac{2\pi}{180} k\theta) + w_{ik} \sin(\frac{2\pi}{180} k\theta)) \sim \mathcal{N}(0, K(\frac{\alpha P}{\sqrt{K}})^2) \sim \mathcal{N}(0, (\alpha P)^2).$$

K should be chosen > 180 to avoid a bias against integer angles, as when $K = 180$ the peaks of the noise are exactly at each integer angle. We use $K = 1800$ as default. As the noise should be the same every time the same angle is put into the function, the v_{ik} and w_{ik} are randomly computed once and stored in a matrix at the start of the computations. Each time data is created, we then use the function $N_i(\theta)$ with the stored values for v_{ik} and w_{ik} to compute the noise for those specific angles. The functions necessary for these operations can be found in appendix C.



(a) No noise (b) Noise 1% (c) Noise 5%

Figure 27: L2 of an image of a rectangle (image size 256) tilted 30 degrees. The picture shows the L2 of a reconstruction of two angles: first (fixed) angle of 30 degrees and the angle on the x-axis.

A.2 The effect of noise on the cost function

In Figure 27, the effect of adding noise to the data on the cost function (See Equation 6), is shown for a tilted rectangle. The first angle is fixed to be the tilt of 30 degrees, the picture shows the value of L for a reconstruction with 2 angles for three different noise levels. As can be seen, adding noise makes the cost function less smooth. There are more local minima, which makes finding the global minima more difficult.

B Image creating functions

B.1 Phantoms

To import this Python file use *import shapes*.

```python
import numpy as np
from scipy import ndimage
import pylab
import math
import printfunctions as prt
from imageio import imread

#%%
def rectangle(size, angle):
    P = np.zeros([size,size])
    P[round(size/5):round(4*size/5),round(size*2/5):(round(size*4/5))] = 1
    Q = ndimage.interpolation.rotate(P,angle, reshape = False,order = 1)
    return Q

def rectangles2(size, angle):
    P = np.zeros([size,size])
    P[round(size/5):round(2*size/5),round(size*2/5):(round(size*4/5))] = 1
    P[round(3*size/5):round(4*size/5),round(size*2/5):(round(size*4/5))] = 1
    Q = ndimage.interpolation.rotate(P,angle, reshape = False, order = 1)
    return Q

def circle(size):
    shape = np.array([size,size])
    xx = np.arange(0, shape[0]) - shape[0] // 2
    yy = np.arange(0, shape[1]) - shape[1] // 2

    r0 = (size/3)**2

    vol = ((xx[:, None, None])**2 + (yy[None, :, None])**2)
    vol = np.array((vol < r0), dtype = 'float32')
    P = np.reshape(vol,(size,size))

    return P

#%%
def boat(size, angle):
    """
    Creates a boat phantom
    size:   the size of the phantom image
    angle:  the tilt of the boat in the image
    """
    parameters = [size/4, 0.02*size, 2]#[outer radius, wall thickness, squeeze]
    shape = np.array([size,size])
    xx = np.arange(0, shape[0]) - shape[0] // 2
    yy = np.arange(0, shape[1]) - shape[1] // 2

    r0 = (parameters[0] - parameters[1])**2
    r1 = (parameters[0])**2

    vol = ((xx[:, None, None]/parameters[2])**2 + (yy[None, :, None])**2)
    vol = np.array(((vol > r0) & (vol < r1)), dtype = 'float32')
    P = np.reshape(vol,(size,size))
    P[int(round(1/5*size)):int(round(4/5*size)),int(round(3/10*size)):
        int(round(7/10*size))]= 0.6
    P[int(round(3/5*size)):int(round(7/10*size)),int(round(3/10*size)):
        int(round(7/10*size))]=1.5
    P[int(round(3/10*size)):int(round(4/10*size)),int(round(3/10*size)):
        int(round(7/10*size))]=1.5

    Q = ndimage.interpolation.rotate(P,angle, reshape = False, order = 1)
    return Q



def dot_256():
    size = 256
    P = np.zeros([size,size])
    P[40:50,40:50] = 1.4
    return P

def phantom2():
    image_data = imread('phantom2.png')
    return image_data

#%%
def bayes_true_image(size, density = 1, printyn = 0, print_angle_pic = 0):
    """
    Creates the true diamond image for the bayes cost function tests.
    """
    a = 0
    b = 0
#    mean_width = size/2
#    mean_heigth = size/3
    mean_width = size/1.6
    mean_heigth = size/5
    total_density = mean_width*mean_heigth*density/2
    width = mean_width+2*a
    heigth = mean_heigth+2*b
    slope = heigth/width
    tan = math.atan(1/slope)
    right_angles = np.array([tan/np.pi*180, (np.pi-tan)/np.pi*180])
    print('Theoretical right angles:', right_angles)

    pix_density = total_density*2/(width*heigth)
    #create a diamond
    xx,yy = np.meshgrid(np.arange(-size/2, size/2), np.arange(-size/2, size/2))
    F = (-yy<-slope*xx+heigth/2) & (-yy<slope*xx+heigth/2) & (-yy>-slope*xx-heigth/2) &
    ↪  (-yy>slope*xx-heigth/2)
    F = pix_density*F
    if printyn ==1:
        pylab.figure()
        pylab.imshow(F,vmin = 0.0, vmax = density*1.5)
        pylab.colorbar()

    if print_angle_pic == 1:
        prt.plot_polar_angles(right_angles, color = "r")

    return F
```

B.2 Diamond generator

To import this Python file use *import generator*.

```
import numpy as np
import pylab

def generator(N, mean_width, mean_heigth, a_range, b_range, size = 256, density = 1, printyn
↪   = 0):
    """
    Generates N dimonds from a distribution with mean width and mean_height
    as specified, with the width varying between mean_width +_a_range
    density is the value of the density of 1 pixel
    size must be at least max(mean_width/2+a_range, mean_heigth/2+b_range)
    """

    np.random.seed(10)

    diamonds = []
    xx,yy = np.meshgrid(np.arange(-size/2, size/2), np.arange(-size/2, size/2))

    for i in range(N):
        a = np.random.uniform(-a_range,a_range)
        b = np.random.uniform(-b_range,b_range)

        total_density = mean_width*mean_heigth*density/2

        width = mean_width+2*a
        heigth = mean_heigth+2*b
        slope = heigth/width

        pix_density = total_density*2/(width*heigth)
        #create a diamond
        z = (-yy<-slope*xx+heigth/2) & (-yy<slope*xx+heigth/2) & (-yy>-slope*xx-heigth/2) &
        ↪  (-yy>slope*xx-heigth/2)
        z = pix_density*z
        diamonds.append(z)

        #code below is for printing the training samples

        if (printyn == 1):
            pylab.figure()
            pylab.imshow(z,vmin = 0.0, vmax = density*1.5)
            pylab.colorbar()
            pylab.savefig('Images/training%i.png'%i, bbox_inches='tight')

    return np.asarray(diamonds)
```

B.3 Training samples

In Figures 28 and 29 the training samples generated by the generator and used in Section 6.2 are shown, both with 30% deviation in the vertical direction and 30% and 0% respectively in the horizontal direction. For these, random seed 10 is used.

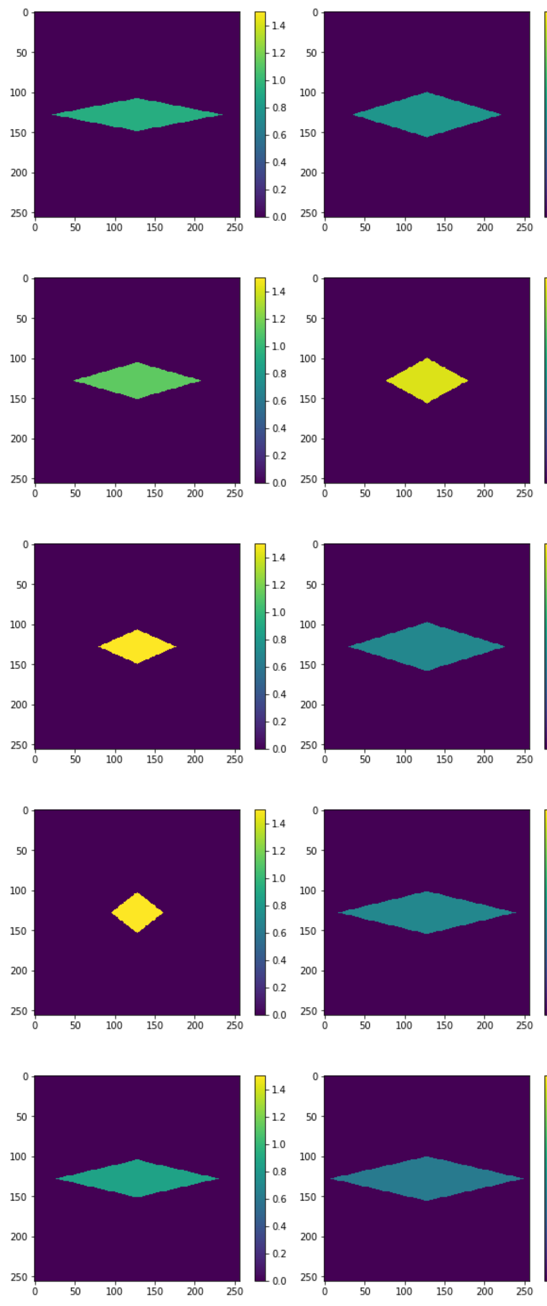


Figure 28: Training samples with 30% variation in x direction, 10% variation in y direction.

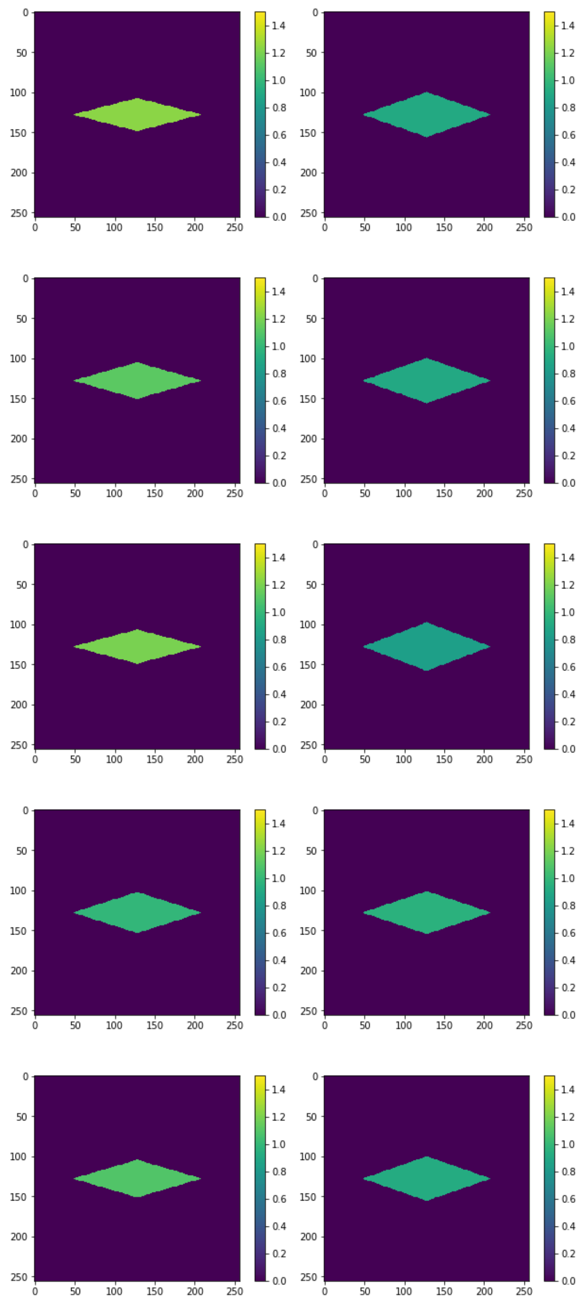


Figure 29: Training samples with 30% variation in the vertical direction and no deviation in the horizontal direction.

B.4 Real data

With these functions the golden standard reconstruction is made from the data given by the FleX-Ray scanner at CWI.

```
import numpy as np
import os
import glob
import skimage.io
import skimage.transform
import matplotlib.pyplot as plt
import re
import astra
from scipy.ndimage import center_of_mass
import datetime

import skimage.filters

"""
The functions below perform the preprocessing steps for real data.
They are adapted from functions by Fredrik Gustafsson, who designed them for the final
↪  project of the Mastermath course Inverse Problems in Imaging.
"""

def get_scan_image_dimension(directory):
    one_scan = glob.glob(directory + '/scan*.tif')[0]
    return skimage.io.imread(one_scan).shape


def load_images(directory, slice_at):
    images = glob.glob(directory + '/scan*.tif')
    # Make sure we process the files in the same order they where scanned.
    images.sort(key=lambda f: int(re.search(r'scan_([0-9]+).tif', f).group(1)))

    first_image = skimage.io.imread(images[0])
    x, y = first_image.shape
    sinogram = np.zeros((len(images), y))

    for i, imagefile in enumerate(images):
        image = skimage.io.imread(imagefile)
        sinogram[i] = image[slice_at]

    def load_averaged_flatfield():
        flatfield_images = glob.glob(directory + 'io*.tif')
        avg_flatfield = np.zeros((1, y))

        for imagefile in flatfield_images:
            image = skimage.io.imread(imagefile)
            avg_flatfield += image[slice_at]

        avg_flatfield = avg_flatfield / len(flatfield_images)
        # Extract middle
        return avg_flatfield

    def load_darkfield():
        return skimage.io.imread(os.path.join(directory, 'di000000.tif'))[slice_at]
```

```python
def preprocess_sinogram(sinogram, avg_flatfield, darkfield, verbose=False):
    # Flat field corrections
    sinogram = np.log((avg_flatfield - darkfield)/(sinogram - darkfield))

    # Center of mass corrections
    row_sum = np.sum(sinogram, axis=0)
    com = center_of_mass(row_sum)[0]

    x, y = sinogram.shape
    true_com = y / 2.0

    if verbose:
        print("Sinogram shape: ", sinogram.shape)

    min_delta = 10000.0
    best_k = -1

    # Assume we're at most 20 pixel away from centor of mass.
    for k in range(1, 20):
        com_k = sinogram[:,:-k].shape[1] / 2.0
        calc_com_k = center_of_mass(sinogram[:,:-k])[1]

        delta = np.abs(com_k - calc_com_k)

        if delta < min_delta:
            min_delta = delta
            best_k = k

        if verbose:
            print("k = {}, com_k = {} calc_com_k = {}, delta = {}".format(
                k, com_k, calc_com_k, delta))

    sinogram = sinogram[:,:-best_k]

    new_row_sum = np.sum(sinogram, axis=0)
    new_com = center_of_mass(new_row_sum)[0]

    if verbose:
        print("True COM ", true_com)
        print("Scan COM", com)
        print("Corrected", new_com)
        print("New True COM", sinogram.shape[1] / 2)
        print("New sino shape", sinogram.shape)

    return sinogram

avg_flatfield = load_averaged_flatfield()
darkfield = load_darkfield()

preprocessed_sinogram = preprocess_sinogram(sinogram[:-1], avg_flatfield, darkfield,
↪   verbose=True)

# Last scan angle == first scan angle, so we drop the last scan
return preprocessed_sinogram, images
```

```
def load_cached_sinogram(filename):
    return np.load(filename)


def reconstruct_image_sirt(proj_angles, sinogram, n_iter, show_reconstruction=False):

    # Warning
    # The geometry is hardcoded for the wooden sample.

    # Scan geometry
    SDD = 498.0
    SOD = 313.001465
    d = 0.149600

    # Apply scaling to match Astras pixelsize v = 1
    v = d * SOD / SDD
    SDD_p = SDD / v
    SOD_p = SOD / v
    ODD_p = SDD_p - SOD_p
    d_p = d / v

    scan_width = sinogram.shape[1]
    vol_geom = astra.create_vol_geom(scan_width, scan_width)
    proj_geom = astra.create_proj_geom('fanflat', d_p, scan_width, proj_angles, SOD_p,
    ↪  ODD_p)
    #proj_id = astra.create_projector('strip_fanflat', proj_geom, vol_geom)
    proj_id = astra.create_projector('cuda', proj_geom, vol_geom)
    W = astra.OpTomo(proj_id)

    reconstruction = W.reconstruct('SIRT_CUDA',
                               sinogram,
                               iterations=n_iter,
                               extraOptions={'MinConstraint': 0.0})

    if show_reconstruction:
        plt.imshow(reconstruction, cmap='gray')
        plt.show()

    return reconstruction



if __name__ == '__main__':

    now = '{date:%Y_%m_%d_%H_%M_%S}.txt'.format( date=datetime.datetime.now() )

    scans_directory = '/export/scratch1/bossema/IPI_Project/wooden_block_upright/'

    x, y = get_scan_image_dimension(scans_directory)

    # Load sinogram from the slice in the middle
    sinogram, images = load_images(scans_directory, slice_at=300)
```

```python
# Only use projection in range [0, \pi)
max_angle = np.pi
sinogram = sinogram[0:sinogram.shape[0]//2]


np.save("wooden_sinogram_vertical.npy", sinogram)


first_image = skimage.io.imread(images[0])


scanned_angles = sinogram.shape[0]
sinogram = sinogram[0:scanned_angles]
scan_width = sinogram.shape[1]

n_iter = 600
proj_angles = np.linspace(0, (scanned_angles-1)*max_angle / scanned_angles,
↪  scanned_angles)

F = reconstruct_image_sirt(proj_angles, sinogram,n_iter,show_reconstruction = True)
np.save("reconstruction_real.npy", F)
```

C Main functions

To import this Python file use *import FB_functions*.

```python
import astra
import numpy as np
import pylab
import scipy
import matplotlib.pyplot as plt

#%%
def plot_polar_angles(angles, x, dot_size = 60, color = "b"):
    """
    Angles in radians
    This functions prints half of the unit circle and the angles as dots
    """

    angles = (angles/np.pi*180)%180

    fig = plt.figure(figsize = (5,5))
    dot_size = 50
    ax = fig.add_subplot(111,projection = 'polar')
    ax.set_yticklabels([])
    ax.set_thetamin(0)
    ax.set_thetamax(180)
    ax.set_theta_offset(0.5*np.pi)
    angles = angles/180*np.pi

    r = np.ones_like(angles)

    ax.scatter(angles, r, s= dot_size, color = color)
    pylab.savefig('Images/angles%d.png'%x, bbox_inches='tight')


def print_image_data(true_image, proj_angles, data, recon_image, proj_size, n_angles, x = 0,
↪   grey = 0):
    """
    This function prints the true image, sinogram and reconstructed image.
    grey:   If 1, images are printed in black and white.
    """
    size = true_image.shape[0]
    if grey == 0:
        pylab.figure()
        pylab.imshow(true_image)
        pylab.colorbar()
        pylab.savefig('Images/true%x.png'%x, bbox_inches='tight')
        if data is not None:
            pylab.figure()
            pylab.imshow(np.reshape(data, (n_angles,proj_size)))
            pylab.colorbar()
        pylab.figure()
        pylab.imshow(np.reshape(recon_image, (size,size)))
        pylab.colorbar()
        pylab.savefig('Images/recon%x.png'%x, bbox_inches='tight')
        plot_polar_angles(proj_angles, x)
    else:
        pylab.figure()
        pylab.imshow(true_image, cmap = 'gray')
        pylab.colorbar()
        pylab.savefig('Images/true%x.png'%x, bbox_inches='tight')
        if data is not None:
            pylab.figure()
            pylab.imshow(np.reshape(data, (n_angles,proj_size)), cmap = 'gray')
            pylab.colorbar()
        pylab.figure()
        pylab.imshow(np.reshape(recon_image, (size,size)), cmap = 'gray')
        pylab.colorbar()
        pylab.savefig('Images/recon%x.png'%x, bbox_inches='tight')
        plot_polar_angles(proj_angles, x)


def equidistant_angles(fixed_angle_nr, first_angle = None):
    """
    This function outputs an array of equidistant angles between 0 and pi.
    fixed_angle_nr:     Total number of angles in the array.
    first_angle:        Angle to put in the array. The other are equidistant around this
 ↪  angle.
    """
    if first_angle is None:
        first_angle = np.random.randint(0,179)/180*np.pi

    angles = np.linspace(first_angle, first_angle + np.pi,
                         fixed_angle_nr,False)
    return angles

def sinogram_noise(F, proj_size, vol_geom, noise_perc = 0.01, total_n_angles = 1800):
    """
    This function creates a sinogram containing only noise.

    """
    proj_angles = np.linspace(0,180,total_n_angles)/180*np.pi
    proj_geom = astra.create_proj_geom('parallel', 1.0, proj_size, proj_angles)
    proj_id = astra.create_projector('strip', proj_geom, vol_geom)
    W = astra.OpTomo(proj_id)

    sino_vector = W.FP(F.ravel())
    noise = noise_perc*sino_vector.max()/(np.sqrt(total_n_angles))
    data = np.zeros(proj_size*total_n_angles)
    noise_sino = np.random.normal(data, noise)
    noise_sino = np.reshape(noise_sino, (total_n_angles,proj_size))
    astra.projector.delete(proj_id)
    return noise_sino


def apply_sino_noise_c(data, proj_angles, proj_size, noise_cos, noise_sin):
    """
    This function adds the noise to the data.
    """
    data = np.reshape(data, (proj_angles.size,proj_size))

    k_vec = np.arange(1,1800+1,1)
    cos_matrix = np.cos(np.outer(proj_angles,k_vec)*2)
    cos_sum = np.dot(cos_matrix,noise_cos)
    sin_matrix = np.sin(np.outer(proj_angles,k_vec)*2)
    sin_sum = np.dot(sin_matrix,noise_sin)

    data = data + cos_sum + sin_sum
    return data.ravel()

#%%
def G_angles_bayes(T, proj_angles, proj_size, vol_geom, noise_cos, noise_sin, n_iter_sirt):
    """
    This function gives the reconstrucion of the images in T.
    Input: list of true images T
    Output: list of reconstructions of T
    """

    proj_geom = astra.create_proj_geom('parallel', 1.0, proj_size, proj_angles)
    proj_id = astra.create_projector('strip', proj_geom, vol_geom)
    W = astra.OpTomo(proj_id)

    n_train = len(T)

    F = T[0]

    data = W.FP(F)
    data = apply_sino_noise_c(data, proj_angles, proj_size, noise_cos, noise_sin)
    T_inv  = W.reconstruct('SIRT_CUDA', data, iterations=n_iter_sirt,
    ↪  extraOptions={'MinConstraint':0.0}).ravel()

    if n_train > 1:
        for i in range(1,n_train):
            Fi = T[i]

            data = W.FP(Fi)
            data = apply_sino_noise_c(data, proj_angles, proj_size, noise_cos, noise_sin)
            f_inv  = W.reconstruct('SIRT_CUDA', data, iterations=n_iter_sirt,
            ↪  extraOptions={'MinConstraint':0.0})
            T_inv = np.concatenate((T_inv,f_inv.ravel()))

    astra.projector.delete(proj_id)

    return T_inv

def L_measure_bayes(angle, T, proj_angles, proj_size,vol_geom, noise_cos, noise_sin,
↪   n_iter_sirt):
    """
    This function calculates the cost function L for an angle added to the existing set of
↪   angles.
    """
    if angle is not None:
        proj_angles = np.append(proj_angles, angle)

    n_train = len(T)

    F_train = T[0].ravel()
    #create the long vector of true images if there are more than 1 training samples
    if n_train > 1:
        for i in range(1,n_train):
            F_train = np.concatenate((F_train,T[i].ravel()))

    T_inv = G_angles_bayes(T, proj_angles, proj_size,vol_geom, noise_cos, noise_sin,
    ↪  n_iter_sirt)

    L2 = 1/n_train*0.5*np.sqrt(((T_inv-F_train)**2).sum())

    return L2

def L_measure_bayes_2angles(x,y, T, proj_size, vol_geom, noise_cos, noise_sin, n_iter_sirt,
↪   scanzero = 0):
    """
    Calculate the L2 value for 2 angles x,y in radians

    x,y:        the two angles
    scanzero:   if 1, the projection angles array always contains 0
    """

    if scanzero == 1:
        proj_angles = np.array([0,x,y])
    else:
        proj_angles = np.array([x,y])

    n_train = len(T)
    F_train = T[0].ravel()
    #create the long vector of true images if there are more than 1 training samples
    if n_train > 1:
        for i in range(1,n_train):
            F_train = np.concatenate((F_train,T[i].ravel()))

    T_inv = G_angles_bayes(T, proj_angles, proj_size, vol_geom, noise_cos, noise_sin,
    ↪  n_iter_sirt)

    L2 = 1/n_train*0.5*np.sqrt(((T_inv-F_train)**2).sum())

    return L2
#%%
def find_min_interval(measure, left_angle, right_angle, T, proj_angles, proj_size, vol_geom,
↪   noise_cos, noise_sin, n_iter_sirt):
    """
    This function find the minimum of the cost function within the interval (left_angle,
↪   right_angle).
    """

    #right and left angle in radians
    if left_angle ==0.0:
        left_angle_degrees = 0
    else:
        left_angle_degrees = round(min(left_angle, right_angle)/np.pi*180-1)

    if right_angle == np.pi:
        right_angle_degrees = 180
    else:
        right_angle_degrees = round(max(left_angle, right_angle)/np.pi*180+1)

    angle_range = np.linspace(left_angle_degrees, right_angle_degrees,
    ↪  round(abs(right_angle_degrees-left_angle_degrees))+1)
    angle_range = angle_range/180*np.pi

    #evaluate points to find a good guess for x min
    x = [measure(i,T, proj_angles, proj_size, vol_geom, noise_cos, noise_sin, n_iter_sirt)
    ↪  for i in angle_range]
    x_min_guess_index = np.where(x==min(x))
    x_min_guess_eval = min(x)
    x_min_guess = angle_range[x_min_guess_index]
    x_min_guess = float(x_min_guess[0])

    #find an angle that is best with the other angles fixed,
    #optimizing measure function, local optimization around the guess
    interval = scipy.optimize.fminbound(measure,x_min_guess-np.pi/180,x_min_guess+np.pi/180,
                                        args=(T,proj_angles, proj_size, vol_geom,
                                        ↪  noise_cos, noise_sin,
                                        ↪  n_iter_sirt),full_output=1,disp = False)
    angle1 = interval[0]
    eval_angle1 = interval[1]

     #Check that the angle found by the optimizer has lower L2 than the angle found by
     ↪  evaluating on the grid
    if eval_angle1 < x_min_guess_eval:
        best_angle_optimizer = angle1
        eval_optimizer = eval_angle1
    else:
        best_angle_optimizer = x_min_guess
        eval_optimizer = x_min_guess_eval


#    print('eval optimizer',eval_optimizer, 'ANGLE', best_angle_optimizer)
#    print('eval current', x_min_guess_eval,'current angle', x_min_guess)
#    print('best angle', best_angle_optimizer)

    return best_angle_optimizer, eval_optimizer
```

C.1 Print functions

To import this Python file use *import printfunctions*.

```
import FB_functions
import numpy as np
import pylab
import matplotlib.pyplot as plt



def print_L2_2angles(T, proj_size, vol_geom, noise_cos, noise_sin, n_iter_sirt = 5, scanzero
↪  = 0, z = 0, scale_max = 1, print_each = 0):
    """
    This function prints the L2 cost function for two angles.
    scanzero:   if 1, the projection angles always contain 0
    z:          the image number (for saving purposes)
    scale_max:  output of print_L2_2angles, with the true image, so that scaling is the same
↪   for each image
    print_each: if 1, the L2 cost function image is printed for each training image
    """
    picsize = 180
    x = np.linspace(0,180,picsize)/180*np.pi
    y = np.linspace(0,180,picsize)/180*np.pi
    X,Y = pylab.meshgrid(x, y)
    X = X.ravel()
    Y = Y.ravel()

    Z = [ (FB_functions.L_measure_bayes_2angles(X[i], Y[i], T, proj_size, vol_geom,
    ↪   noise_cos,noise_sin, n_iter_sirt, scanzero) if X[i]<Y[i] else 0) for i in
    ↪   range(len(X))]
    Z = np.reshape(Z,(picsize,picsize))
    if T.shape[0] == 1:
        A = Z[Z>0]
        Z = (Z-A.min())
        scale_max = Z.max()
    Z = Z/scale_max


    if print_each == 1:
        for j in range(len(T)):
            F = T[j]
        #Print each L2 function
            pylab.figure()
            image = pylab.imshow(F, vmin = 0.0, vmax = 1.5)
            pylab.colorbar(image)
            pylab.xlabel('x')
            pylab.ylabel('y')

            pylab.figure(figsize = (13,13))
            im = pylab.imshow(Z,cmap=pylab.cm.RdBu, vmin = 0.0, vmax = 1.5)
            plt.contour(Z, levels=np.linspace(0,1,10+1),colors='white')

            pylab.colorbar(im)
            pylab.xlabel('angle 1')
            pylab.ylabel('angle 2')
            plt.xlim(0,180)
            plt.ylim(0,180)
            plt.show()


    min_notzero = Z[Z>Z.min()].min()
    minima = np.asarray(np.where(Z==min_notzero))
    minimum = [(minima[:,i] if minima[0,i] > minima[1,i] else None) for i in
    ↪   range(minima.shape[1])]
    minimum = [x for x in minimum if x is not None][0]
    print('The minimum of the L2 costfunction average is at:', minimum)

    pylab.figure(figsize = (13,13))
    im = pylab.imshow(Z,cmap=pylab.cm.RdBu, vmin = 0.0, vmax = 1.5)
    plt.contour(Z, levels=np.linspace(0,1,10+1),colors='white')
    pylab.colorbar(im)
    plt.plot(minimum[1],minimum[0], 'yo', ms = 10)
    pylab.xlabel('angle 1')
    pylab.ylabel('angle 2')
    pylab.title('average L2')
    plt.xlim(0,180)
    plt.ylim(0,180)

    pylab.savefig('Images_bayes/L2_%x.png'%z, bbox_inches='tight')
    plt.show()

    if T.shape[0]:
        return Z, scale_max
    else:
        return Z


#%%

def add_arrow(line, position=None, size=30, color=None):
    """
    add an arrow to a line.

    line:       Line2D object
    position:   x-position of the arrow. If None, mean of xdata is taken
    size:       size of the arrow in fontsize points
    color:      if None, line color is taken.
    """
    if color is None:
        color = line.get_color()

    xdata = line.get_xdata()
    ydata = line.get_ydata()

    if position is None:
        xposition = xdata.mean()
        yposition = ydata.mean()

    start_ind = 0

    line.axes.annotate('',
        xytext=(xdata[start_ind], ydata[start_ind]),
        xy=(xposition,yposition),
        arrowprops=dict(arrowstyle="->", color=color),
        size=size
    )



def plot_L2_particles_2(Z,ensemble_tracker, lines = 0, only_last = 1):
    """
    Plot the track of the particles of the ensemble Kalman filter
    Z:          the L2 function that forms the background
    ensemble_tracker: a nested matrix containing the ensemble at each iteration
    lines:      if 1, a line with an arrow connects the old position and the new position of
↪   the last updated particles
    only_last:  if 1, the function only prints the final ensemble
    """
    ensemble_tracker = ensemble_tracker/np.pi*180
    n_particles = len(ensemble_tracker[0][0,:])
    n_iter = len(ensemble_tracker)-1
    n_angles =len(ensemble_tracker[0][:,0])

    colorvec = ['bo','go','ro', 'co', 'mo', 'yo', 'ko', 'wo',
                'b^','g^','r^', 'c^', 'm^', 'y^', 'k^', 'w^',
                'bs','gs','rs', 'cs', 'ms', 'ys', 'ks', 'ws',
                'bP','gP','rP', 'cP', 'mP', 'yP', 'kP', 'wP',
                'b*','g*','r*', 'c*', 'm*', 'y*', 'k*', 'w*']
    colorvec2 = ['b-','g-','r-', 'c-', 'm-', 'y-', 'k-', 'w-']

    fig, ax = plt.subplots(figsize=(13, 13))

    im = ax.imshow(Z,cmap=pylab.cm.RdBu, vmin = 0)
    plt.xlim(0,180)
    plt.ylim(0,180)
    plt.xlabel('angle1')
    plt.ylabel('angle2')
    plt.colorbar(im)
    for j in range(n_particles):
            particle_track = np.zeros([n_angles, n_iter+1])
            for i in range(n_iter+1):
                particle_track[: , i] = ensemble_tracker[i][:,j]
            if only_last == 1:
                ax.plot(particle_track[0,n_iter],particle_track[1,n_iter],
                    colorvec[j%len(colorvec)], ms = 10)
            else:
                ax.plot(particle_track[0,:],particle_track[1,:],
                    colorvec[j%len(colorvec)], ms = 10)
            if lines == 1:
                last_update = particle_track[:,n_iter-1:n_iter+1]
                ax.plot(last_update[0,:],last_update[1,:],
                        colorvec[j%len(colorvec)], ms =10)
                line = ax.plot(last_update[0,:],last_update[1,:],
                        colorvec2[j%len(colorvec2)], ms =10)[0]
                add_arrow(line)

    plt.show()
```

D Angle selection algorithms

D.1 Greedy algorithm

To import this Python file use *import bayes_angle_optimization_greedy*.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 14 10:41:37 2018

@author: bossema
"""

import numpy as np
import astra
import FB_functions
import printfunctions as prt


def angle_opt_greedy(T, proj_size, vol_geom, noise_cos, noise_sin, n_iter_sirt,
↪  fixed_angle_nr,
                 first_angle = None):
    """
    #In this function we try to optimize the cost function for a fixed number
    #of projection angles.
    T:                  set of real images
    fixed_angle_nr:     the total number of projection angles
    first_angle:        If given, this angle is in the initial angle set.
    """

    if first_angle is None:
        first_angle = np.random.randint(0,179)/180*np.pi

    #If a fixed angle number is given, the program will find that number of
    #angles

    proj_angles = np.array(first_angle)
    left_angle = 0.0
    right_angle = np.pi

    for j in range(fixed_angle_nr-1):
    #In this for-loop we find in each iteration the best angle to add to the
    #ones we have already chosen in previous iterations, based on the
    #L2 norm for each angle added to the existing set
        best_angle = FB_functions.find_min_interval(FB_functions.L_measure_bayes,
        ↪  left_angle, right_angle, T, proj_angles, proj_size, vol_geom, noise_cos,
        ↪  noise_sin, n_iter_sirt)[0]
        proj_angles = np.append(proj_angles,best_angle)

    #Print which angles were chosen

    proj_angles = np.sort(proj_angles)
    #print('Optimal angles greedy:', proj_angles)
    return proj_angles
```

D.2 Coordinate descent algorithm

To import this Python file use *import bayes_angle_optimization_cdescent*.

```
import FB_functions
import numpy as np


def angle_update(angle_index,T, proj_angles, proj_size, vol_geom, noise_cos, noise_sin,
↪  n_iter_sirt):
    """
    This function updates the angle for the coordinate descent method. Angles in radians.
    """


    if angle_index == 0:
        left_angle = 1
    else:
        left_angle = proj_angles[angle_index - 1]

    if angle_index == proj_angles.size - 1:
        right_angle = np.pi
    else:
        right_angle = proj_angles[(angle_index+1)]

    #delete the angle at the current index
    current_angle = proj_angles[angle_index]
    proj_angles = np.delete(proj_angles, angle_index)
    current_angle_eval = FB_functions.L_measure_bayes(current_angle, T, proj_angles,
    ↪  proj_size, vol_geom, noise_cos, noise_sin, n_iter_sirt)


    #Find the minimum of the cost function on the given interval
    best_angle_optimizer, eval_optimizer =
    ↪  FB_functions.find_min_interval(FB_functions.L_measure_bayes, left_angle,
    ↪  right_angle, T, proj_angles, proj_size, vol_geom, noise_cos, noise_sin, n_iter_sirt)


    #Check that the newly found angle has lower evaluation than the current angle
    if eval_optimizer < current_angle_eval:
        best_angle = best_angle_optimizer
    else:
        best_angle = current_angle


    #Insert the best angle at the current index
    proj_angles = np.insert(proj_angles, angle_index, best_angle)
    return proj_angles


def angle_opt_cdes(T, proj_size, vol_geom, noise_cos, noise_sin, n_iter_sirt,
↪  fixed_angle_nr,
                first_angle = None, initial_angles = None,
                n_iter = None, stop = None):
    """
    T:                  set of real images
    fixed_angle_nr:     the total number of projection angles
    first_angle:        If given, this angle is in the initial angle set.
    initial_angles:     An initial guess for all initial angles, if not given
                        the program will take equidistant initial angles
    n_iter:             number of iterations of the update step
    stop:               stopping criterium for the difference between the old angles and the
↪
                        updated ones
    Either specify n_iter or stop, if none is provided, the function takes the
    default stopping value.
    Angles in radians.

    """

    if first_angle is None:
        first_angle = np.random.randint(0,179)/180*np.pi


    if initial_angles is None:
        proj_angles = FB_functions.equidistant_angles(fixed_angle_nr,
                                                       first_angle)
    else:
        proj_angles = initial_angles

    proj_angles = np.sort(proj_angles%np.pi)


    if n_iter is None:
        if stop is None:
            stop = np.pi/360 #stopping criterium is if the angles do not change
            #more than a half degree

    #update angles subject to a maximum number of iterations
    if n_iter is not None:
        for k in range(n_iter):#number of iterations of all angle update
            for i in range(fixed_angle_nr):#update all the angles once
                proj_angles = angle_update(i,T, proj_angles, proj_size, vol_geom, noise_cos,
                 ↪  noise_sin, n_iter_sirt)#update step



#Code for using a stopping criterium instead of a fixed number of iterations
    if stop != None: #update the angles subject to a stopping criterium
        max_diff = stop + 1
        k = 0
        while max_diff > stop: #continue updating untill the max difference
            #between an angle and its update is smaller than the stopping
            #criterium
            k = k+1
            difference = np.zeros(proj_angles.size) #keep track of difference
            for i in range(fixed_angle_nr):
                old_angle = proj_angles[i]
                proj_angles[i] = angle_update(i,proj_angles,T,noise_cos, noise_sin,
                           proj_size,vol_geom,first_angle,n_iter_sirt) #update step
                difference[i] = abs(proj_angles[i]-old_angle)
            max_diff = max(difference)
        print('Number of iterations c_des untill stopping criterium was reached:',
              k)


    #print('Optimized angles after coordinate descent:', proj_angles)
    return proj_angles
```

D.3 Ensemble Kalman algorithm

To import this Python file use *import bayes_angle_optimization_ensemble*.

```
import FB_functions
import numpy as np


def project_part(part):
    """This function makes sure that the first angle is smaller than second etc
    and that if a particle is outside the upper triangle of the image, it is backprojected
↪   to the nearest edge of the region taking off the edges.
    Thus, any angle between -90 and 1 is backprojected to 1 and any angle between 179 and
↪   270 is backprojected to 179
    """
    for i in range(len(part)):
        if (part[i]%(2*np.pi)>270/180*np.pi) or (0>part[i]%(2*np.pi)<1/180*np.pi):
            part[i] = 1/180*np.pi
        if np.pi-1/180*np.pi<part[i]<270/180*np.pi:
            part[i] = np.pi-1/180*np.pi
    part = np.sort(part)

    return part


def angle_opt_ens(T,ensemble, proj_size, vol_geom, noise_cos, noise_sin, n_iter_sirt = 5,
↪   n_iter = 10, update_nr = 0, stepsizes = None):
    """
    This function optimizes the set of angles starting out from a set of
    angles, all angles in degrees. a list of arrays
    T:                  set of real images
    ensemble:           some (randomized) number of particles to start with,
                        a matrix of size number of angles times number of particles, in
↪   radians
    stepsizes:          the stepsizes corresponding to each iteration (size n_iter)
    n_iter:             the number of update steps of the ensemble
    update_nr:          the number of particles to update in every iteration of the
↪   algorithm
    """

    if stepsizes == None:
        stepsizes = np.ones(n_iter)

    nE = ensemble.shape[1]

    for i in range(nE):
```

```
    ensemble[:,i] = np.sort(ensemble[:,i])


n_train = len(T)
m = T[0].ravel().size*n_train


F = T[0].ravel()
#create the long vector of true images if there are more than 1 training samples
if n_train > 1:
    for i in range(1,n_train):
        F = np.concatenate((F,T[i].ravel()))


#number of particles
nAng = ensemble.shape[0]
GE = np.zeros([m,nE])
L2_matrix = np.zeros([n_iter+1,nE])
#matrix to store the data in for different particles of size #nE times length of the
↪  data
ensemble_tracker = [None]*(n_iter+1)
ensemble_tracker[0] = ensemble



#initialize the GE matrix
for j in range(nE):
    GE[:,j] = FB_functions.G_angles_bayes(T, ensemble[:,j], proj_size, vol_geom,
    ↪  noise_cos, noise_sin, n_iter_sirt)
    L2_matrix[0,j] = ((GE[:,j]-F)**2).sum()

#update the angles n_iter times
for i in range(n_iter):

    #calculate the fixed elements of the update step
    E_mean = np.mean(ensemble, axis = 1)
    E_mean = np.reshape(E_mean,[nAng,1])
    GE_mean = np.mean(GE, axis = 1)
    GE_mean = np.reshape(GE_mean,[m,1])
    Juw = 1/np.sqrt(nE-1)*(ensemble-E_mean)
    Jww = 1/np.sqrt(nE-1)*(GE-GE_mean)
    gamma_inv = 1/stepsizes[i]
    inv_term = np.linalg.inv(np.identity(nE)+gamma_inv*(Jww.T@Jww))
    L2_matrix[i+1,:] = L2_matrix[i,:]

    #If no number of particles to update is given (update_nr), we will update all the
    ↪  particles every step.
    if update_nr == 0:
        update_particles = range(nE)
    else: #if a update_nr is given, the worst (i.e. highest diff) update_nr particles
    ↪  will be updated
        update_particles = np.argpartition(L2_matrix[i,:],-update_nr)[-update_nr:]
    update_particles = np.sort(update_particles)

    for j in update_particles: #update particles
```

```
            diff = F-GE[:,j]
            ensemble[:,j] = ensemble[:,j] + Juw@(Jww.T@(gamma_inv*diff -
            ↪  gamma_inv*Jww@(inv_term@(Jww.T@(gamma_inv*diff)))))
            #Here the sort needs to be changed: if a particle gets out of range, how to
            ↪  project it back into the range
            ensemble[:,j] = project_part(ensemble[:,j]) #make sure all angles are sorted and
            ↪  within 0 and 180 degrees.
            GE[:,j] = FB_functions.G_angles_bayes(T, ensemble[:,j], proj_size, vol_geom,
            ↪  noise_cos, noise_sin, n_iter_sirt) #update the GE evaluation
            L2_matrix[i+1,j] = ((GE[:,j]-F.ravel())**2).sum()

        ensemble_tracker[i+1] = ensemble

    particle_nr = np.argmin(L2_matrix[n_iter,:])
    particle = ensemble[:,particle_nr]
#    print('The optimal angles found by the EnsembleK Filter are:',particle)
#    print('Number of iterations:',n_iter)
#    print('Number of particles updated per iteration:',update_nr)


    return (particle, ensemble_tracker, ensemble)
```

E Test files

E.1 Selection algorithms

```python
import shapes
import bayes_angle_optimization_greedy as angle_greedy
import astra
import numpy as np
import bayes_angle_optimization_cdescent as angle_cdes
import bayes_angle_optimization_ensemble as angle_ens
import FB_functions
import pylab
from generator import generator
import printfunctions as prt
import skimage
import matplotlib.pyplot as plt

#Here the different variables can be modified
tilt = 90
noise_perc = 0.0
fixed_angle_nr = 10
first_angle = tilt/180*np.pi
n_iter_cdes = 10
update_nr = 2
n_iter_sirt =100
n_iter_sirt_def = n_iter_sirt
scanzero = 0
size = 256
proj_size = (3*size)//2

np.random.seed(10)
mean_width = size/1.6
mean_heigth = size/5
density = 1

#%%
image_nr = 2

grey_on = 1
#%%#For testing with the real image define a real image F
#F = shapes.rectangle(size,tilt)
#F = shapes.circle(size)
#F = shapes.boat(size, tilt)
#F = shapes.bayes_true_image(size)

#%%For testing with the image based on real data
F = np.load("/ufs/bossema/Code python/Test2D/realdata/reconstruction_real.npy")
size = F.shape[0]
proj_size = (3*size)//2

T = np.array([F])


#%%
#For equidistant angles
#proj_angles = FB_functions.equidistant_angles(10,90/180*np.pi)
```

```
#fixed_angle_nr = proj_angles.size
#%%
#To test a set of training samples T
#np.random.seed(10)
#T = generator(10, mean_width, mean_heigth, 0.*mean_width,mean_heigth*0.3, size = size,
↪  density = density, printyn = 0)
#F = shapes.bayes_true_image(size, density = 1, print_angle_pic = 0)
#%%
vol_geom = astra.create_vol_geom(size, size)
noise_cos = FB_functions.sinogram_noise(F,proj_size,vol_geom, noise_perc)
noise_sin = FB_functions.sinogram_noise(F,proj_size,vol_geom, noise_perc)

#%%
#To test either the greedy algorithm or the coordinate descent algorithm choose
#one of the two definitions of proj_angles below, or use both to perform
#a coordinate descent after the greedy algorithm

#proj_angles= angle_greedy.angle_opt_greedy(T, proj_size, vol_geom, noise_cos, noise_sin,
↪  n_iter_sirt, fixed_angle_nr, first_angle)

#proj_angles = angle_cdes.angle_opt_cdes(T, proj_size, vol_geom, noise_cos, noise_sin,
↪  n_iter_sirt, fixed_angle_nr, initial_angles = proj_angles, n_iter = n_iter_cdes)
#

#%%
#To test the Ensemble Kalman filter use the code below

#nE = 10
#n_iter_ens = 10
#np.random.seed(10)
#ensemble1 = np.random.rand(fixed_angle_nr,nE)*np.pi
#ensemble_filter = angle_ens.angle_opt_ens(T,ensemble1, proj_size, vol_geom, noise_cos,
↪  noise_sin, n_iter_sirt= n_iter_sirt, n_iter = n_iter_ens, update_nr = update_nr)
#proj_angles = ensemble_filter[0]


#ensemble_tracker = ensemble_filter[1]
#ensemble2 = ensemble_filter[2]
#print('Ensemble after filter method:',ensemble2)

#%%
#The code below is needed for using the L2 particles plot function (ensemble method only)
##Calculate L2
#picsize = 180
#x = np.linspace(0,179,picsize)
#y = np.linspace(0,179,picsize)
#X,Y = pylab.meshgrid(x,y)
#X = X.ravel()
#Y = Y.ravel()
#
#
#Z = [(FB_functions.L_measure_bayes_2angles(X[i], Y[i], T, proj_size, vol_geom, noise_cos,
↪  noise_sin, n_iter_sirt, scanzero) if X[i]<=Y[i] else 0) for i in range(len(X))]
#Z = np.reshape(Z,(picsize,picsize))
#A = Z[Z>0]
```

```
#Z = (Z-A.min())
#scale_max = Z.max()
#Z = Z/scale_max
#fig, ax = plt.subplots(figsize=(13, 13))
#
#im = ax.imshow(Z,cmap=pylab.cm.RdBu, vmin = 0)
#plt.contour(Z, levels=np.linspace(0,1,10+1),colors='white')
#plt.xlim(0,180)
#plt.ylim(0,180)
#plt.xlabel('angle1')
#plt.ylabel('angle2')
#plt.colorbar(im)
#plt.show()
#
#%%
##To print all the particles and their movements

#prt.plot_L2_particles_2(Z,ensemble_tracker, lines = 0, only_last = 1)



#%%

f_inv  = FB_functions.G_angles_bayes(T, proj_angles, proj_size, vol_geom, noise_cos,
↪  noise_sin, n_iter_sirt)
print('Angles:', proj_angles/np.pi*180)

data = None
FB_functions.print_image_data(F, proj_angles, data, f_inv, proj_size, fixed_angle_nr,
↪  image_nr, grey = grey_on)


angle = None
L2= FB_functions.L_measure_bayes(angle, T, proj_angles, proj_size,vol_geom, noise_cos,
↪  noise_sin, n_iter_sirt)
print('The L2 of this reconstruction:', L2)

#To print the difference between the reconstruction and the phantom

#f_inv = np.reshape(f_inv, (960,960))
#pylab.imshow(F-f_inv, cmap = 'gray', vmin = -0.0025, vmax = 0.0025)
#pylab.colorbar()
#pylab.show()
```

E.2 Comparison figure

```
import shapes
import bayes_angle_optimization_greedy as angle_greedy
import astra
import numpy as np
import bayes_angle_optimization_cdescent as angle_cdes
import bayes_angle_optimization_ensemble as angle_ens
import FB_functions
import pylab
from generator import generator
```

```
import printfunctions as prt
import skimage
import matplotlib.pyplot as plt




#%%
def angle_experiment(X,F, first_angle, n_iter_sirt, noise_perc, method):


    n_iter_cdes = 10

    update_nr = 2
    nE = 10*len(X)
    n_iter_ens = 10

    T = np.array([F])

    size = F.shape[0]
    proj_size = (3*size)//2
    vol_geom = astra.create_vol_geom(size, size)
    noise_cos = FB_functions.sinogram_noise(F,proj_size,vol_geom, noise_perc)
    noise_sin = FB_functions.sinogram_noise(F,proj_size,vol_geom, noise_perc)


    L2_total = np.array([])

    for i in X:

        fixed_angle_nr = i

        if method == 'equi':
            proj_angles = FB_functions.equidistant_angles(fixed_angle_nr,tilt)

        if method == 'greedy' or method == 'coordinate':
            proj_angles= angle_greedy.angle_opt_greedy(T, proj_size, vol_geom, noise_cos,
            ↪  noise_sin, n_iter_sirt, fixed_angle_nr, first_angle)

        if method == 'coordinate':
            proj_angles = angle_cdes.angle_opt_cdes(T, proj_size, vol_geom, noise_cos,
            ↪  noise_sin, n_iter_sirt, fixed_angle_nr, initial_angles = proj_angles, n_iter
            ↪  = n_iter_cdes)

        if method == 'ensemble':
            np.random.seed(10)
            ensemble1 = np.random.rand(fixed_angle_nr,nE)*np.pi
            ensemble_filter = angle_ens.angle_opt_ens(T,ensemble1, proj_size, vol_geom,
            ↪  noise_cos, noise_sin, n_iter_sirt= n_iter_sirt, n_iter = n_iter_ens,
            ↪  update_nr = update_nr)
            proj_angles = ensemble_filter[0]

        angle = None
        L2= FB_functions.L_measure_bayes(angle, T, proj_angles, proj_size,vol_geom,
        ↪  noise_cos, noise_sin, n_iter_sirt)
        #print('The L2 of this reconstruction:', L2)
        #print('Angles:', proj_angles)

        #data = None
        #FB_functions.print_image_data(F, proj_angles, data, f_inv, proj_size,
        ↪  fixed_angle_nr, image_nr, grey = grey_on)

#        Z = F-np.reshape(f_inv, (size,size))
#        Z = Z
#        pylab.figure()
#        pylab.imshow(Z, cmap = 'gray')
#        pylab.colorbar()
#        pylab.show()

        L2_total = np.append(L2_total,L2)


    return L2_total

#%%
size = 256
tilt = 30
Flist = [shapes.rectangle(size,tilt), shapes.circle(size), shapes.boat(size, tilt),
↪  shapes.boat(size, tilt) + np.roll(np.roll(shapes.boat(size, 68),20, axis=0),10,axis =
↪  1), np.load("/ufs/bossema/Code python/Test2D/realdata/reconstruction_real.npy")]
names = ['Rectangle','Circle','Boat','Doubleboat','Wood']
n_iter_sirt_list = np.array([5,5,5,5,100])
tilt_list = np.array([30,30,30,30,90])/180*np.pi
noise_perc_list = np.array([0.01, 0.05])
noise_list = np.array([1,5])
X = np.array([2,3,4,5,10,15,20])
for j in range(len(noise_perc_list)):
    noise_perc = noise_perc_list[j]
    for i in range(len(Flist)-1):
        F = Flist[i]
        n_iter_sirt = n_iter_sirt_list[i]
        tilt = tilt_list[i]
        L2_equi = angle_experiment(X,F,tilt,n_iter_sirt, noise_perc,'equi')
        L2_ens = angle_experiment(X,F,tilt,n_iter_sirt,noise_perc,'ensemble')
        L2_greedy = angle_experiment(X,F,tilt,n_iter_sirt,noise_perc,'greedy')
        L2_coordinate = angle_experiment(X,F,tilt,n_iter_sirt,noise_perc,'coordinate')

        plt.figure()
        plt.plot(X,L2_equi, label = 'Equidistant')
        plt.plot(X,L2_ens, label = 'Ensemble')
        plt.plot(X,L2_greedy, label = 'Greedy')
        plt.plot(X,L2_coordinate, label = 'Coordinate')
        plt.legend(loc='upper right')
        plt.xticks(X)
        plt.ylabel('L')
        plt.xlabel('Number of angles')
        plt.savefig('/ufs/bossema/Dropbox/Scriptie/Grafieken/%s'%names[i] +
        ↪  '_noise%x'%noise_list[j] + '.png', bbox_inches='tight')
        plt.show()
```

E.3 Cost function for Bayesian approach

```
import FB_functions
import numpy as np
import astra
from generator import generator
from shapes import bayes_true_image
import printfunctions
import matplotlib.pyplot as plt


noise_perc = 0.0
size = 256
proj_size = (3*size)//2
mean_width = size/1.6
mean_heigth = size/5
density = 1
n_iter_sirt = 5


F = bayes_true_image(size)
T = np.array([F])

vol_geom = astra.create_vol_geom(size, size)
noise_cos = FB_functions.sinogram_noise(F,proj_size,vol_geom, noise_perc)
noise_sin = FB_functions.sinogram_noise(F,proj_size,vol_geom, noise_perc)

Z,scale_max = printfunctions.print_L2_2angles(T,proj_size, vol_geom, noise_cos, noise_sin,
↪   n_iter_sirt, scanzero = 1, z = 0, scale_max = 1, print_each = 0)

T = generator(10, mean_width, mean_heigth, 0.0*mean_width, mean_heigth*0.1, size = size,
↪   density = density, printyn = 1)
AvL2 = printfunctions.print_L2_2angles(T,proj_size, vol_geom, noise_cos, noise_sin,
↪   n_iter_sirt, scanzero = 1, z = 0, scale_max = scale_max, print_each = 0)



T2 = generator(10, mean_width, mean_heigth, 0.3*mean_width, mean_heigth*0.1, size = size,
↪   density = density, printyn = 1)
TL20,AvL20 = printfunctions.print_L2_2angles_trainset_each(T2,proj_size, vol_geom,
↪   noise_cos, noise_sin, n_iter_sirt, scanzero = 1, z = 3, scale_max = scale_max,
↪   print_each = 0)
```