UNIVERSITY OF COPENHAGEN

MSc in Computational Physics

Improving performance by guiding Markov Chain Monte Carlo sampling

The purpose is to use computational, statistical and numerical methods to improve the performance of a Monte Carlo algorithm. The informed proposal Monte Carlo algorithm will be used to solve inverse problems.

Jacob Henriksen

Supervised by Klaus Mosegaard

20th. May 2022

Jacob Henriksen

Improving performance by guiding Markov Chain Monte Carlo sampling

MSc in Computational Physics, 20th. May 2022

Supervisor: Klaus Mosegaard

University of Copenhagen

Niels Bohr Institute

Section for the Physics of Ice, Climate and Earth

Tagensvej 16

2200 Copenhagen N

0. Acknowledgements

I would like to thank Klaus Mosegaard for his inspiration, discussions and guidance throughout the process. Furthermore, I would like to thank the people at PICE for creating a good working atmosphere. Finally, I would like to thank my fellow students at the PICE office.

0. Abstract

In this thesis, I combine probabilistic inverse problems with Markov Chain Monte Carlo algorithms and informed proposal Monte Carlo algorithms to estimate the posterior probability distribution of two different experiments. The first experiment I investigate is a box experiment, where I consider gravity data from a buried box, which I sample with a Markov Chain Monte Carlo, and an Informed proposal Monte Carlo. The second experiment is an acoustic wave reflected on a wall experiment, where I combine both methods once again to sample the thickness of two alternating layers inside the wall. Both experiments are supported by clear representations to obtain overview of the the content. I generate all observed data by knowing the forward function and the model parameters, and I sample the model parameters with inverse problem theory. Due to the modelization error, the informed proposal algorithm performs the sampling incredibly faster, compared to the blind Markov Chain Monte Carlo algorithm. In the box experiment, the Informed proposal Monte Carlo algorithm performs the sampling with 200 times less iterations compared to the Markov Chain Monte Carlo, and in the acoustic wave experiment the Informed proposal performs the sampling with 1000-1500 times less iterations compared to the Markov Chain Monte Carlo.

Contents

1. Introduction

As there is an increasing interest and progress in the development of algorithms in technology, scientists must be innovative in speeding up the advancement of an algorithm. One of the things that can contribute to this process is the implementation of informed algorithms. Compared to blind algorithms, informed algorithms have proven to perform incredibly faster due to the information an informed algorithm is given.

Ever since the discovery of a Monte Carlo algorithm, it was realized that sampling algorithms could lead to fantastic scientific developments within research of inverse problems and geophysics, for example, reservoir characterization investigation for creating geological models. with the purpose of gaining all the knowledge available about the parameters of the earth, the probabilistic inverse problems are used and the uncertainty analysis is performed.

For now the informed algorithms are a field of computational physics which attracts vast efforts and investments since it would enable us to solve more complex problems much faster, and it would minimize the delay in technological instruments using the algorithm.

In the daily use of an informed proposal algorithm, more intelligent technology could be developed. For example, instruments using an acoustic wave towards a building wall to measure the distance between the layers inside the wall with no delay in obtaining the results. This experiment has been investigated in this thesis, and furthermore, an experiment imitating a box structure inside the earth has been investigating. Both experiments have been solved with a blind Markov Chain Monte Carlo algorithm and an Informed Proposal Algorithm. Finally, the results are compared and the speed up process has been investigated.

2. Inverse Problems

The way of working in geophysics is sometimes opposite from the classical way of working in physics. Especially, when inner earth physics is investigated. This means that we in geophysics sometimes need to describe the earth with mathematics in an inverted way, and it can be either inversion or parameterization. In this process, we wish to obtain a group of model parameters which we call $\mathbf{m}$. The model parameters $\mathbf{m}$ are obtained through a function f. E.g the function f describes a mapping of the structure of the earth. This leads to a very simple equation.

$$\mathbf{m} = f(m) \tag{2.1}$$

Furthermore, it is important to remember that data is hugely important for any investigation in physics. Data is a set of numbers that describes the connection between our physical system, our measuring tool, and the way we choose to measure. Data used in inverse problem is called the observed data since it comes from an observation. Our observation is what is used to find our desired model parameters. Since the model parameters is not measurable, they are strongly desired to estimate, and it is done through inverse problem theory. This leads to the formulation of the inverse problem which can be described as:

$$\mathbf{d}_{obs} = g(\mathbf{m}) \tag{2.2}$$

Where $\mathbf{d}_{obs}$ is the observed data, $\mathbf{m}$ is the model parameters and g is the transformation operator, which transforms the data from the model space to the data space. Furthermore, it is desired to estimate the model parameters $\mathbf{m}$.
Two ways of working with inverse problems are highly used. The fist one is the deterministic approach and the second one is the probabilistic approach. The probabilistic approach is briefly explained in this section, and further explained later in section 3.3, when sampling the pobability density is explained. In deterministic approaches, the minimization between the observed data and the result obtained from theoretical calculations are used. from the minimization one final model is produced, and often it describes the "*best achievable model*". The deterministic approach has limitations, and the limitations are due to the non-uniqueness of the solution obtained and due to sensitivity to possible errors. Examples of this, could be regularization methods

such as Tikonov regularization.

In most geophysical experiments, we encounter non-linear problems. However, linear inverse problem also occur. For this thesis, the non-linear problems will be the main focus. This leads to a statistical approach for estimating the model parameters. From statistics, the most powerful equation is the Bayes Theorem, and it describes the probabilities of an event which can be updated by giving the occurrence of a rel event. This could be described as the probability of event m giving the event d:

$$f(\mathbf{m}|\mathbf{d}) = \frac{f(\mathbf{d}|\mathbf{m})f(\mathbf{m})}{f(\mathbf{d})} \tag{2.3}$$

Where $f(m|d)$ is the posterior, $f(d|m)$ is the likelihood and $f(m)$ is the prior. The definition of conjunction of information can be used to state that in the joint space $D \times M$ we get:

$$\sigma(\mathbf{d}, \mathbf{m}) = (\rho \vee \theta)(\mathbf{d}, \mathbf{m}) = \frac{\rho(\mathbf{d}, \mathbf{m})\theta(\mathbf{d}, \mathbf{m})}{\mu(\mathbf{d}, \mathbf{m})} \tag{2.4}$$

Equation 2.4 is the Information Theory Formulation described by Tarantola-Valette in 1982 (Tarantola, 1982). Furthermore, it is evident that $\sigma(\mathbf{d}, \mathbf{m})$ is the posteori, $\rho(\mathbf{d}, \mathbf{m})$ is the prior and $\theta(\mathbf{d}, \mathbf{m})$ is the forward density function. Additionally, $\mu(\mathbf{d}, \mathbf{m})$ is the null information density function. Now we can use the assumption that $\mu(\mathbf{d}, \mathbf{m}) = \mu(\mathbf{d})\mu(\mathbf{m})$ which states that μ, as the homogeneous probability distribution, denotes a probability proportional to the volume of each region $\mathbf{d}$ and $\mathbf{m}$. Furthermore, the solution to the inverse problem can be defined as the probability density function and the likelihood function. By knowing that, the posterior probability distribution is obtained.

$$\sigma(\mathbf{m}) = k\rho_m(\mathbf{m})L(\mathbf{m}) \tag{2.5}$$

The likelihood function $L(\mathbf{m})$ describes the connection between model parameters and observed data and will be further explained in the implementation section 3.2.

2.1 Blind Algorithms

In general, a lot of algorithms can be used for mathematical simulations and sampling. Examples of this could be the Monte Carlo, Genetic algorithms, etc. These types of algorithms can be defined as blind algorithms. This is due to the characteristics of the target probability distribution which we do not consider in the process of constructing the algorithm and the sampling process. This means that the blind algorithm just considers the input and the output data from an optimization

"black box", and no additional information about the target distribution is provided to the algorithm (Khoshkholgh *et al.*, 2021b). The blind algorithm is the opposite of an informed proposal algorithm, which is going to be used and constructed for this thesis.

2.2 No-free-Lunch Theorem

Two types of algorithms can be defined based on the information they are provided. The two types are called blind algorithms and informed algorithms.
Additionally, information about the blind algorithm can be added in terms of a formulation based on the usage of an oracle. An oracle is a type of function that is used to evaluate the target distribution f at a given point x. The blind algorithm uses a more heuristic method to solve problems since the oracle is used as a "black box" for the algorithmic search. Examples of blind algorithms are Genetic algorithms, Neural networks and the simple type of MCMC algorithms.
As mentioned earlier, the other type of algorithm is an informed algorithm. An informed algorithm also uses an oracle, however, the oracle uses known external properties of the function f to guide or improve the sampling from the function f.
The No-Free-Lunch theorem states that the average efficiency of all blind optimization algorithms is exactly the same for all optimization problems. This means that a certain algorithm can be more effective for a specific problem, however it will not be more effective for other problems. This is due to the reason that the average efficiency of all optimization algorithms is similar when dealing with all available problems. Additionally, the No-Free-Lunch theorem states that if an informed algorithm is used considering features of the desired target distributing it will be more effective than blind algorithms (Mosegaard, 2012).
The Informed algorithm is the topic of investigation in this thesis, and more specifically, the informed proposal algorithm is desired to construct and use for performance investigation. The main purpose is to investigate if the informed proposal algorithm uses less iterations than a blind MCMC algorithm. All steps used to create the transition from a blind MCMC algorithm to a informed proposal algorithm will be explained and the needed approximations will be introduced and shown graphically.

3. Monte Carlo methods

The Monte Carlo method is a simple technique used to solve problems with high complexity. The name of the Monte Carlo method is often associated with the randomness from the casino houses in the district of Monte Carlo in Monaco. This association is not truly incorrect since the Monte Carlo method is based on randomness, for example, randomness in gambling.
A more specific definition is that the Monte Carlo method is a random evolving process that samples what we call the probability density. This sampling is recognized by performing a random walk in the desired space. This means that the Monte Carlo process is a numerical process that produces pseudo random numbers. These pseudo random numbers are series of numbers that appear randomly if they are tested with a statistical test. A central part of the Monte Carlo process is the generation of pseudo-random numbers which in most cases are used as a uniformly distribution in the interval from 0 to 1. This process transforms the pseudo random numbers into pseudo random samples (Sambridge and Mosegaard, 2002).
For inverse problems, we might encounter a situation where the physical relation between model space and data space is non-linear. This means that the posterior probability density function is non-Gaussian, and the posterior probability density function that cannot be estimated from a linearization method. A Monte Carlo method is preferable to use to obtain satisfying results for this case. (Sambridge and Mosegaard, 2002). Additionally, the use of the Monte Carlo method will provide uncertainty information for the posterior probability density function which can be used to estimate the model parameters of the inverse problem.

3.1 Markov Chain and Burn in phase

To find feasible solutions for the posterior distribution, we need simulation based methods. This is due to the intractable behavior of the posterior distribution. To do this, Markov Chain Monte Carlo (MCMC) can be a good choice of simulation. The Basic idea behind the implementation of the MCMC is an interplay between proposals and rejections.
The Markov chain process uses a random walk process since the probability of moving from a point x_i to a point x_j in the space χ in a given step is independent of the

precious path travelled. This leads to the definition of conditional probability distribution $P_{ij}(x_i|x_j)$ of visiting x_i given that the previous point visited was x_j (Sambridge and Mosegaard, 2002).

One of the most important properties of the MCMC simulation is to take the burn-in phase into account. This means that often we need to discard some of the initial states of the simulation since these states are not representative of the desired distribution needed for the sampling. This means a large amount of the first iterations can be useless results. This is due to the fact that the simulation chain has not reached the desired stationary distribution. This is, as mentioned before, the burn-in period of the Markov Chain Monte Carlo. For some physical problems, this type of simulation can be time demanding and computationally expensive. The interest and motivation of this thesis is to investigate if the burn-in phase can be minimized as much as possible by gaining information from mathematical steps and "give" that information to the algorithm. This is what we call the informed proposal Monte Carlo, and in this type of algorithm the Markov Chain process is eliminated from the sampling process.

3.2 Implementation of MCMC for an inverse problem

The implementation of the MCMC sampler contains a probabilistic formulation in order to construct a posteriori probability density. The posteriori is the prior multiplied with the likelihood (Mosegaard and Tarantola, 1995).

Posteriori probability density

$$\sigma(\mathbf{m}) = \frac{\rho(\mathbf{m})\, L(\mathbf{m})}{\mu(\mathbf{m})} \tag{3.1}$$

A priori probability density

$$\rho(\mathbf{m}) = const\ \exp\left\{\left(-\frac{1}{2}(\mathbf{m}-\mathbf{m}_0)^T C_m^{-1}(\mathbf{m}-\mathbf{m}_0)\right)\right\}, \tag{3.2}$$

Likelihood function

$$L(\mathbf{m}) = const\ \exp\left\{\left(-\frac{1}{2}(\mathbf{d}_{obs}-g(\mathbf{m}))^T C_d^{-1}(\mathbf{d}_{obs}-g(\mathbf{m}))\right)\right\} \tag{3.3}$$

Both co-variance matrices C_m and C_d are assumed to be diagonal with a standard deviation respectively. The *homogeneous probability density function* $\mu(\mathbf{m})$ is assumed to be constant and therefore not implemented as it would cancel out in calculation

of acceptance probabilities. Furthermore, instead of using the $\rho(\mathbf{m})$ and $L(\mathbf{m})$ as they are defined above, the implementation uses the logarithms of both functions to increase numerical stability.

Algorithm 1 Basic MCMC algorithm For an inverse problem

1: *Choose*: $\mathbf{m}_0$ and set $\mathbf{m}_0$=$\mathbf{m}_{cur}$
2: *Perturb*: $\mathbf{m}_{per} = \mathbf{m}_{cur} + (2u-1)step$
3: *Compute*: $\rho(\mathbf{m}_{cur})$ and $\rho(\mathbf{m}_{per})$
4: *Compute*: $L(\mathbf{m}_{cur})$ and $L(\mathbf{m}_{per})$
5: *Compute*: $p_{acc} = min(1, \frac{\rho(\mathbf{m}_{pert})L(\mathbf{m}_{pert})}{\rho(\mathbf{m}_{curr})L(\mathbf{m}_{curr})})$
6: *Generate*: Random numbers u: $0 \leq u \leq 1$
7: **if** $u \leq P_{accept}$ **then**
8: accept m_{per}
9: **end if**
10: **if** $u > P_{accept}$ **then**
11: Reject m_{per}
12: **end if**
13: return to 2

By looking at algorithm 1, it is possible to write up the way of sampling the posterior model distribution. According to algorithm 1, we start by choosing a starting model $\mathbf{m}_0$, and we define $\mathbf{m}_{curr}$ which is the current model parameter opposite to the perturbed model parameter $\mathbf{m}_{pert}$. The basic idea is that $\mathbf{m}_{curr}$ is perturbed by replacing one parameter in the current model. That parameter is chosen as a randomly proposed value or as a sequentially chosen value. For the basic example, the sampling is done by generating a uniformly random distributed value u. For the basic example, this value is chosen between 0 and 1. This leads to the principle of the perturbation:

$$\mathbf{m}_{pert} = \mathbf{m}_{curr} + (2u-1)\mathbf{e}_{pert} \tag{3.4}$$

where $\mathbf{e}_{pert}$ is the direction of perturbation expressed as a unit vector. $\mathbf{e}_{pert}$ denotes that only one direction coordinate is perturbed at a time. We remember that if u is a random number between 0 and 1 then $2u-1$ is a random number between -1 and 1. Now $\rho(\mathbf{m})$ and $L(\mathbf{m})$ is calculated as shown in equation 3.2 and 3.3. This is done to calculate the acceptance probability which is given from the Metropolis criteria:

$$p_{acc} = min(1, \frac{\rho(\mathbf{m}_{pert})L(\mathbf{m}_{pert})}{\rho(\mathbf{m}_{curr})L(\mathbf{m}_{curr})}) \tag{3.5}$$

The next step is to calculate a random number u between 0 and 1 in order to make the algorithm decide if the perturbation of the current model is accepted or rejected. This means that $\mathbf{m}_{pert}$ is accepted if $u <= p_{acc}$ and $\mathbf{m}_{curr} = \mathbf{m}_{pert}$. Furthermore, $\mathbf{m}_{pert}$ is rejected if $u > p_{acc}$ and the algorithm starts over, and the $\mathbf{m}_{curr}$ is used during

the next iteration once again. In general, an acceptance rate between 30-70 percent accepted models is a well performing MCMC algorithm (Mosegaard and Sambridge, 2002).
Additionally, it is important to remember that it is numerically preferable to work with logarithms of probability functions or likelihood functions. This is due to the possibility of getting a results too close to zero that we risk by computational round off . This can result in making the algorithm move around in low probability areas, unable to know the desired direction. This means that the actual implementation is done with the logarithm of the likelihood function, and then calculate the exponential function of the acceptance probability according to equation 3.5. When the algorithm is done running, a sequence of vectors are generated, and the idea is that the numbers generated in the vector are oscillating around some average after cutting the burn-in phase away. We remember that the output are solutions that fit the data within the error bars. Additionally, it produces samples from the posteriori probability distributions $\sigma(\mathbf{m}) = \rho(\mathbf{m})L(\mathbf{m})$. We remember that the sampled solutions are the probability distribution which can be characterized by computing properties from the sample.

3.3 Sampling the probability density

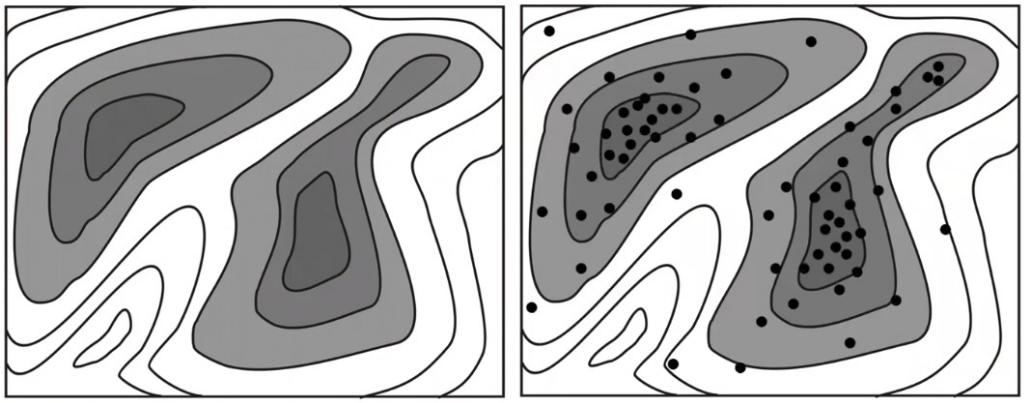

Figure 3.1: Basic principle of the sampling with a probability density (Mosegaard, 2006).

By looking at figure 3.1, a 2D space representation of two peaks are shown. The area of the two peaks are a probability density assigning higher probability to the dark area. The posterior probability is the desired probability according to formula 3.1. The posterior probability is the final output estimated from the probabilistic inversion scheme. The posterior probability is estimated with a Monte Carlo algorithm since no closed formula solution exits for these non-linear inverse problems. Instead an algorithm is built that visits points in the space with a density proportional to the

probability density. By looking at figure 3.1, a representation of the sampling is shown according to the two probability densities. These sampling methods are called the probabilistic methods in contrast to the deterministic method. As mentioned earlier, the solution to the probabilistic method is a sampling of the posterior. An advantage of the Monte Carlo method is that the method will not search for only one peak but for solutions/samples throughout the desired area, since the probability density is used to decide the area of the sampling. Figure 3.2 shows the sampling of

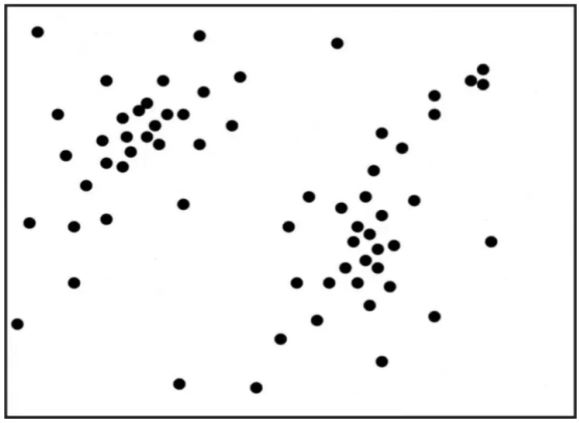

Figure 3.2: Basic principle of the sampling with a probability density. The points represent the sampled probability density as a multi-modal target distribution. (Mosegaard, 2006).

the posterior probability density as generated points proportional to the probability density. The samples generated are information about the probability density, and it can be used for uncertainty analysis. Mostly, integrals can be used in the model space to find properties of the sampling. For the case in figure 3.1 and figure 3.2, a multi-modal distribution is evident. For a multi-modal distribution, an expectation of the samples is not very informative. However, other integral based statistical analysis could contribute to obtaining useful information about the posterior.

4. Informed Proposal

4.1 Using information derived from an approximate forward relation

Initially, we consider the general expression for the joint posterior probability in the formulation of Tarantola and Valette (1982) (Tarantola, 1982):

$$\sigma\left(\mathbf{d},\mathbf{m}\right)=\frac{\rho\left(\mathbf{d},\mathbf{m}\right)L\left(\mathbf{d},\mathbf{m}\right)}{\mu\left(\mathbf{d},\mathbf{m}\right)} \tag{4.1}$$

In this equation $\mathbf{d}$ is the data, and $\mathbf{m}$ are the model parameters. $\rho\left(\mathbf{d},\mathbf{m}\right)$ is the prior probability density, and $\mu\left(\mathbf{d},\mathbf{m}\right)$ is the homogeneous probability density. The expression $\theta(\mathbf{d},\mathbf{m})$ describes the "uncertainty of the forward relation" between $\mathbf{m}$ and $\mathbf{d}$. Now it is possible to assume that the homogeneous probability density $\mu(\mathbf{d},\mathbf{m})$ as well as the marginal prior in the model space $\rho_{\mathbf{m}}(\mathbf{m})$ are constant. By making this assumption, we can write an expression for the joint posterior.

$$\sigma(\mathbf{d},\mathbf{m})=k\rho(\mathbf{d})\theta(\mathbf{d},\mathbf{m}) \tag{4.2}$$

In this equation, k is the normalization constant. Furthermore, it is assumed that the observed data has a small uncertaincy compared to the modelization errors. Additionally, it should be remembered that, at small data uncertainties, we obtain MCMC algorithms showing a critical slowing-down. Now we can write up an approximation to the posterior in the model space

$$\sigma_m(\mathbf{m})\propto\sigma(\mathbf{d}_{obs},\mathbf{m})\approx\theta(\mathbf{d}_{obs}\mathbf{m}) \tag{4.3}$$

Equation 4.3 is a rough approximation to the posterior in the model space. This approximation will be used to speed up the sampling of the MCMC algorithm to obtain the true posterior. Now the basic idea is to use the solution to the inverse problem with simplified physics according to the section of the approximate forward model. We call the approximate model $\mathbf{m}$. The deviation of this from the true solution is what we call the (true) modelization error $\delta\mathbf{m}_{true}$. This modelization error is desired since, if obtained, we can use it to build the desired modelization error

distribution $\theta(\mathbf{d}_{obs}, \mathbf{m})$. Now the steps for creating the informed proposal MCMC wil be stated. We must remember that we do not know the true solution for the real data inverse problem. Furthermore, the calculation of the error $\delta\mathbf{m}_{true}$ is not possible since $\mathbf{m}_{true}$ in unknown. We remember the idea of using information derived model from an approximate forward relation. Additionally, we wish to create an artificial inverse problem that is an approximation to the original problem. For this approximation, we can write up the approximate modelization error $\delta\mathbf{m}_{approx}$. The main point here is that we construct the approximate problem close to the real problem which makes δm_{approx} close to $\delta\mathbf{m}_{true}$.

- Simplified forward model The intention is to create a simplified forward model $\tilde{g}(\mathbf{m})$. The forward model expresses most of the relevant physics. In this case, the simplified forward model is a box constructed from a sphere with estimated gravity anomely based on a traditional gravitation model equation 6.3.

- "Pseudo inverse" In this step the pseudo inverse h is used to estimate the solution $\tilde{\mathbf{m}} = h(\mathbf{d}_{obs})$. $\tilde{\mathbf{m}}$ is the simplified model with acceptable data fit. Furthermore, the pseudo inverse h must give a unique answer. This unique answer can be obtained from a regularization technique.

- Modelization error The modelization error is estimated by using $\tilde{g}(\mathbf{m})$ instead of using the the regular forward function $g(\mathbf{m})$. The distribution $\theta(\mathbf{d}_{obs}, \mathbf{m})$ is used to estimate the modelization error. We remember that this error is an approximation to the posterior $\sigma_{\mathbf{m}}(\mathbf{m})$. We define the true modelization error as

$$\delta\mathbf{m}_{true} = \tilde{\mathbf{m}} - \mathbf{m}_{true} \tag{4.4}$$

  As mentioned earlier, m_{true} is unknown so we can rewrite equation 4.4 to

$$\delta\mathbf{m}_{true} = h(g(\tilde{\mathbf{m}})) - \tilde{\mathbf{m}} \tag{4.5}$$

  Equation 4.5 states a trivial but important property of the modelization error. It states that the equation estimates what the modelization is if $\tilde{\mathbf{m}}$ had been the true model. Furthermore, if $\tilde{\mathbf{m}}$ is close to $\mathbf{m}_{true}$, it must be reasonable to state that $\delta\mathbf{m}_{approx}$ will be close to $\delta\mathbf{m}_{true}$.

- New proposal distribution Now a new approximate modelization error distribution is obtained and used as proposal distribution.

$$q(\mathbf{m'}|\mathbf{m}) = \theta(\mathbf{d}_{obs}, \mathbf{m}) \tag{4.6}$$

When the probability function q is obtained, the informed proposal can be provided to the algorithm. The function q will be explained later when the implementation of the information derived from an approximate forward relation is made (Khoshkholgh *et al.*, 2021b) (Khoshkholgh *et al.*, 2021a).

4.2 Sampling with informed proposal

An informed proposal Monte Carlo algorithm is useful as it reduces the number of iterations by using mathematical steps to gain useful information for the basic Monte Carlo. In principle, highly non-linear problems with a high number of model parameters can be solved with the cost of only a few number of extra iterations. As mentioned earlier, the first and second order approximations in algorithm 2 refer to the modelization error.

Algorithm 2 Implementation of IPMC algorithm

1: *Estimate*: $\mathbf{m}_0$ as the first order approximation.
2: *Estimate*: $\mathbf{m}_1$ as the second order approximation.
3: *Compute*: u as a random number between -1 and 1.
4: *Compute*: $\mathbf{m}_{pert} = \mathbf{m}_0 \cdot 3 \cdot u \cdot std(\mathbf{m}_0, \mathbf{m}_1)$.
5: *Compute*: $p_{acc} = \frac{\sigma(\mathbf{m}_{pert})}{\sigma(\mathbf{m}_{curr})} \frac{q(\mathbf{m}_{curr})}{q(\mathbf{m}_{pert})}$.
6: *Generate*: Random numbers u: $0 \leq u \leq 1$.
7: **if** $u \leq P_{accept}$ **then**
8: accept $\mathbf{m}_{per}$
9: **end if**
10: **if** $u > P_{accept}$ **then**
11: Reject $\mathbf{m}_{per}$
12: **end if**
13: return to 3

By looking at figure 4.1, the true posterior σ is shown to the left and the informed proposal q is shown to the right. The name of q is the proposal distribution. Typically the proposal distribution is given by the following notation as expression 4.7. The expression denotes the probability of taking a step to point $\mathbf{m}_{n+1}$ given the point $\mathbf{m}_n$ is the current point which means the probability of making a perturbation.

$$q(\mathbf{m}_{n+1}|\mathbf{m}_n) \tag{4.7}$$

The informed proposal is an approximation to the true posterior. In most cases, the informed proposal is a normal distribution centered in the first order approximation with a width corresponding to three times the standard deviation between the first and the second order approximation. Three times the standard deviation is only used for this thesis. Normally, it would be considered as a reduction of guidance

when the multiplication of a natural number is multiplied with the perturbation. The dimensions of the normal distribution corresponds to the number of model parameters that the experiment contains. The proposal distribution is the equation that proposes new steps to sample without bias, as shown i figure 4.1. Furthermore, q is explained in equation 4.6. The equation q can be written as:

$$q(\mathbf{m}_n) = kexp(\frac{1}{2}((\mathbf{m}_n) - \mathbf{m}^1)\mathbf{C}^{-1}((\mathbf{m}_n) - \mathbf{m}^1) \tag{4.8}$$

This leads to an extension of the acceptance probability. In a basic MCMC algorithm,

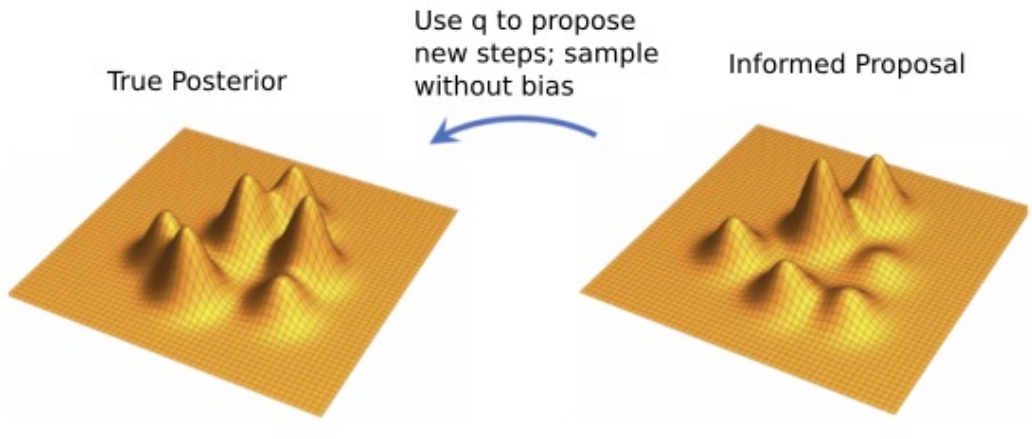

Figure 4.1: A true posterior and an approximate informed proposal is shown. The proposal q is used to the acceptance probability of sample without bias (Mosegaard).

the acceptance probability is estimated from the posterior of the current and the perturbed model. However, for the informed proposal Monte Carlo, the modelization error is calculated as equation 4.8 . The new acceptance probability is given by the following equation.

$$p_{acc} = \frac{\sigma(\mathbf{m}_{n+1})}{\sigma(\mathbf{m}_n)} \frac{q(\mathbf{m}_n|\mathbf{m}_{n+1})}{q(\mathbf{m}_{n+1}\mathbf{m}_n)} \tag{4.9}$$

An important difference between the IPMC and the MCMC algorithm is that, in the MCMC algorithm, each proposal is dependent on the last since it is a Markov Chain process. However, in the IPMC algorithm, each proposal is independent of the last step. This leads to the following reduction for the IPMC algorithm.

$$p_{acc} = \frac{\sigma(\mathbf{m}_{n+1})}{\sigma(\mathbf{m}_n)} \frac{q(\mathbf{m}_n)}{q(\mathbf{m}_{n+1})} \tag{4.10}$$

Multiplying with the proposal distribution q in equation 4.10 secures no bias to the sampling since q is used to point where the IPMC is supposed to sample. In the actual implementation of the proposal distribution q, the logarithm is used to secure numerical stability. This leads to an even more simple expression with the

difference of the current model and the perturbed model. The detailed explanation of this implementation is explained in the relevant section. One of the most important properties of this acceptance probability is that it guarantees that in the limit where the number of models $N \to \infty$, the posterior distribution $\sigma(\mathbf{m})$ will be correctly sampled. Another main property of the proposal distribution is that the choice of proposal is fairly optional. Most important, is that equation 4.10 is well defined for all perturbed models. However, no knowledge of σ is known before the sampling which limits the choice of q.

5. Box Experiment

The first relevant example to study is the gravity box experiment. This experiment is constructed by analysing the gravity anomaly over a box. The gravity anomaly is used as observed data for the Monte Carlo algorithm later with the purpose of estimating the desired model parameters for the box.

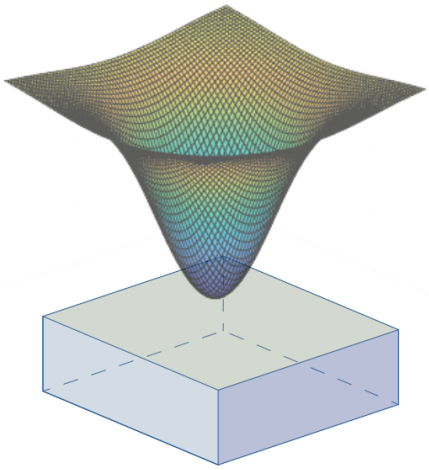

Figure 5.1: This figure shows the principle of the gravity anomaly over a buried box. The box imitates a reservoir desired for characterization, and the 3D curve shows the observed data. Adobe illustrator is used to generate this representation.

The idea of the forward function of this experiment is to consider the gravitational attraction of a prism(Bongiolo *et al.*, 2013). The principle of Newton's law of attraction between two bodies is used since the two masses are inversely proportional to the square of the difference between them. Instead of considering the masses, the corner points of the prism can be used to estimate the gravity anomaly of the prism. The gravity anomaly can be estimated according to the formula 5.1 as the integral over vertical components of the potential field. Furthermore, a_i,b_j and z_k are the corner points of the squared box used for gravity estimation.

$$g_z = \gamma\rho \int_{a1}^{a2} \int_{b1}^{b2} \int_{z1}^{z2} \frac{z dz dy dx}{r^3} \tag{5.1}$$

This can be rewritten to the desired forward function as a discrete triple sum

$$g_z = \gamma\rho \sum_{a1}^{a2} \sum_{b1}^{b2} \sum_{z1}^{z2} s(z_k \tan^{-1} \frac{a_i b_i}{z_k R_{ijk}} - a_i ln(R_{ijk} + b_i) - b_i ln(R_{ijk} + a_i)) \tag{5.2}$$

R_{ijk} is an array including all distances from each grid point generated to the corner point of the prism and is given by:

$$R_{ijk} = \sqrt{a_i^2 + b_j^2 + z_k^2} \tag{5.3}$$

and s is an array consisting of -1 and 1.
For this example, only two transverse points are constructed, since two transverse points are coordinate representative for all eight corner points of a box. The discrete gravity anomaly can be estimated as the sum of the edges of the box with distance R_{ijk} as the distance from the measure grid point to the corner point of the box.

5.1 Non-linearity of the Inverse Problem

Since the natural logarithmic part of equation 5.2 can be simplified as follows:

$$\Delta g_j \propto \sum_{i=0}^{N_x} \ln\left[f_i\left(\mathbf{x}, \mathbf{m}|j\right)\right] = \ln\left[\prod_{i=0}^{N_x} f_i\left(\mathbf{x}, \mathbf{m}|j\right)\right] \equiv \ln\left[f^*(\mathbf{x}, \mathbf{m})\right], \tag{5.4}$$

with $f^*(\mathbf{x}, \mathbf{m})$ being a rational function. Trivially, the natural logarithm of a rational function is non-linear. Furthermore, Δg is proportional to a part in the equation containing the inverse tangent function. We remember that the inverse tangent function is non-linear.

5.2 Markov Chain Monte Carlo sampling

Before the Monte Carlo method is used on the box experiment, we wish to decide the properties of the desired box. To keep the experiment simple, all side lengths are of the same size. The six model parameters are used for generating the observed data according to equation 5.2. However, after the generation of the observed data, the six model parameters are kept unknown for the MCMC algorithm later. Two vector representing of all six model parameters can be constructed according to two transverse points of the box:

$$\mathbf{m}_1 = \begin{pmatrix} -2.54 \\ 2.54 \\ -2 \end{pmatrix} \tag{5.5}$$

$$\mathbf{m}_2 = \begin{pmatrix} 2.54 \\ -2.54 \\ -7.08 \end{pmatrix} \tag{5.6}$$

The sampling is done without a prior. This means that the sampling is done with only a likelihood function as given in equation 3.3. Herewith the posterior consists of only the likelihood function and a constant prior equal to 1. The six model parameters are estimated by running the MCMC sampling for 40,000 iterations. It means that the two x, the two y and the two z coordinates of the box are estimated, from uncertainty analysis of the posterior, by giving the algorithm the gravity anomaly data as the observed data. The sampling is done with 40,000 iteration and a step size of 0.3. The acceptance rate is around 62 percent as denoted in figure 5.4. The perturbation is done for one model parameter per iteration. The model parameters perturbed are chosen randomly due to the concept of figure 5.2. Two parameters are created; Random Perturbation and Random Index. The parameter, Random Perturbation can go from 1 to 3, and the parameter Random Index can go from 1 to 2. In that way, 6 different outcomes are possible and chosen randomly. The random outcome decides which model parameter is perturbed. The Python code made for producing the simulation is shown in appendix section 12.1.

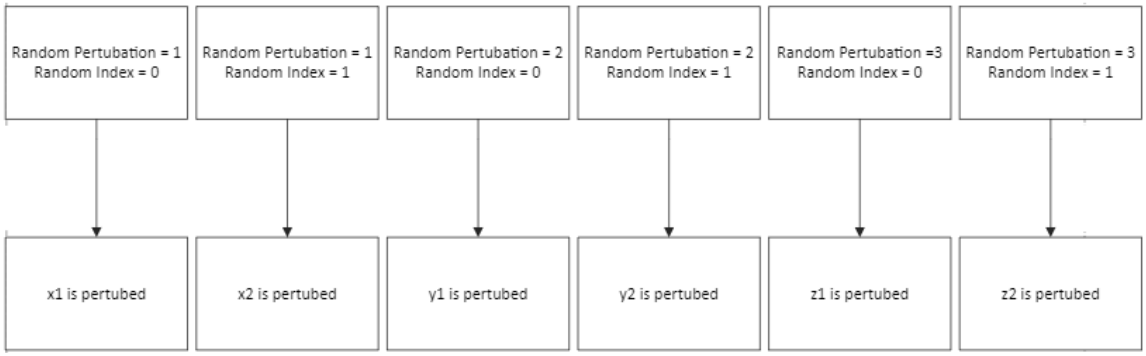

Figure 5.2: A flow chart for the possible outcomes chosen randomly. The outcome decides which model parameter is perturbed. The two x, the two y and the two z parameters are collected into 3 vectors. That is why 3 different random perturbations are chosen with its reverse of the same letter together. This makes the random index 2.

By looking at figure 5.3 and figure 5.4, it is evident that the MCMC simulation was done successfully. The histograms for the different parameters show a decent distribution around the desired model parameters. The standard deviation is fairly small, too. Furthermore, it is evident that MCMC algorithm is challenged to solve the specific depth of the box, since a more spread out distribution is shown for the depth parameter (the purple one). The reason is that a change in the depth does not affect the misfit in the likelihood function as much as a change in width of the box. A more narrow distribution for the x and y is evident compared to the distribution of the depth. By looking further at figure 5.4, a clear burn-in phase is shown on a normal and on a logarithmic scale of the iterations. The logarithmic scale is made to get a more clear view of the burn-in phase.

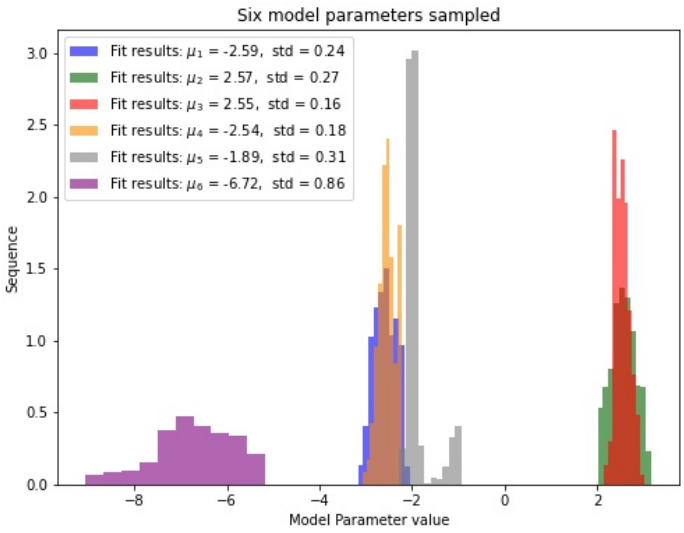

Figure 5.3: The six model parameter distributions sampled with a Markov Chain Monte Carlo (MCMC) algorithm. The mean and standard deviation is denoted in the legend for each model parameter sampled. μ_1 and μ_2 correspond to x_1 and x_2. Furthermore, μ_3 and μ_4 correspond to y_1 and y_2. Furthermore, μ_5 and μ_6 correspond to z_1 and z_2

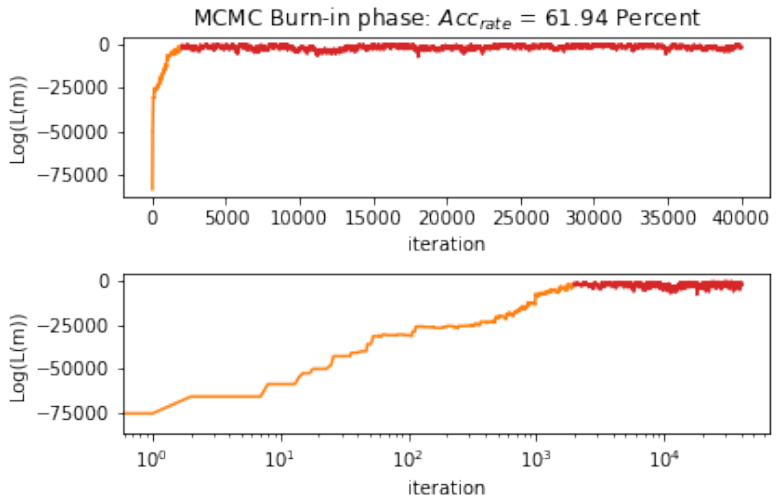

Figure 5.4: The curve shows convergence towards equilibrium for the box experiment. A Markov Chain Monte Carlo (MCMC) method is used with 6 model parameters, which means all the model parameters of the box. The orange part of the curve shows the burn-in phase, and the red part shows the iterations after equilibrium. The MCMC was constructed with a Gaussian proposal perturbing one parameter at a time.

6. Box Experiment with Informed Proposal

Now the MCMC algorithm is successfully implemented according to algorithm 1, and it was shown that it was able to estimate all the model parameters for the box example from the sampled posterior. Now we wish to improve the performance of the MCMC algorithm by implementing a guided version of the MCMC algorithm. The guided version of the MCMC algorithm is called an informed proposal algorithm, and it is based on the modelization error explained in chapter 4. We consider the same box as before:

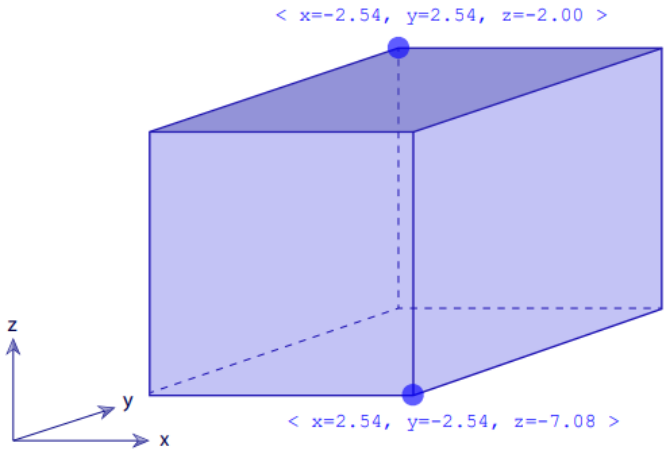

Figure 6.1: This figure shows the "true-unknown" box that we wish to sample with an informed proposal algorithm. Adobe illustrator has been used for these box representations.

Figure 6.1 shows an illustration of the box that we wish to use for estimating the model parameters. The center of the box is placed in $c =< x = 0, y = 0, z = -4.504 >$. To estimate the model parateres of this box with an informed proposal algorithm, we wish to use the theory of an approximate forward relation.

6.1 An approximate forward relation to the box experiment

In order to implement an approximate forward relation, we consider point mass in order to use a modelization error technique. The point mass example is inspired by a classical geophysical problem about finding the depth to the top of a salt dome under the surface layer of the earth. The idea is that the gravity anomaly can be used to estimate the distance from the surface of the earth to the salt dome. Instead of using a salt dome, we use a point mass inside our constructed box and finally the anomaly is used to create a modelization error that can be used as an informed proposal to the Marco Chain Monte Carlo algorithm.

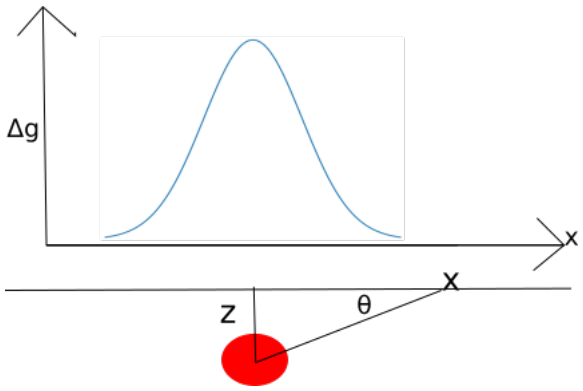

Figure 6.2: This figure shows the gravity anomaly over a point mass. Density difference between the point mass and the surroundings is positive, shown by the upwards pointing gravity anomaly. This figure is generated in paint.

Figure (6.2) shows how a point mass contributes to a gravity anomaly. The point mass is placed with a depth z under the surface and a distance x to the measuring point. The angle between the x line and the point mass is given by θ. By drawing a rectangular triangle, the gravity anomaly can be estimated. The gravity anomaly is pointing upwards if the density difference between the point mass and the surroundings is positive, and the gravity anomaly is pointing downwards if the difference is negative. We know that the general formula for the gravity contribution by a point mass is given by $\Delta g = \frac{Gm}{r^2}$. Now we note that the x axis is placed with an angle θ to the point mass which gives:

$$\Delta g = \frac{Gm}{r^2} sin(\theta) \tag{6.1}$$

We can rewrite $sin(\theta) = \frac{z}{r}$

$$\Delta g = \frac{Gm}{r^2} \frac{z}{r} \tag{6.2}$$

Furthermore, r^2 can be written as $r^2 = x^2 + z^2$ and we know that $m = v\Delta\rho$. That gives:

$$\Delta g_z = \frac{G}{r^2}\frac{4}{3}\pi\Delta\rho R^3\frac{z}{r} = \frac{4}{3}\pi G\Delta\rho R^3\frac{z}{r^3} \tag{6.3}$$

Now we know that $r^3 = (x^2 + z^2)^{3/2}$

$$\Delta g_x = \frac{4}{3}\pi G\Delta\rho R^3\frac{z}{(z^2 + x^2)^{3/2}} \tag{6.4}$$

By looking at the formula above it is evident that Δg will have its maxima or minimum when x is zero according to a PDF distributed around the point mass. Now we consider a box with equal side length in SI units according to figure 6.1. This figure shows the true box with unknown model parameters. Once again the model parameters of the box are sampled with the earlier implemented MCMC algorithm. A Gaussian fit on the accepted model parameters is used to estimate the mean of each model parameter. The minima of the gravity anomaly is noted and used for further calculations. The minimum of the gravity anomaly is estimated to $3.89 * 10^{-10}\ m/s^2$. Now it is possible to obtain the radius in equation 6.4.

$$\Delta g = -3.89 * 10^{-10} m/s^2 = \frac{4}{3}\pi G\Delta\rho R^3\frac{z}{(z^2 + 0^2)^{3/2}} => R = 2.32m \tag{6.5}$$

The radius of point mass is estimated to $R = 2.32m$. Forthermore the radius of the point mass can be used to estimate the mass of the point mass. This is done by using the classical mass estimation formula used for classical mechanics.

$$m_{sphere} = 4/3\pi 2.32m^3(-1kg/m^3) = -52.49kg. \tag{6.6}$$

The mass is estimated to 52.49 kg. Now it is possible to estimate the volume of the sphere and assume that the volume of the sphere is the same as the volume of the first order approximation to the box.

$$v_{sphere} = v_{box} = \frac{m_{sphere}}{\rho_{sphere}} = \frac{-52.49kg}{-1kg/m^3} = 52.49m^3 \tag{6.7}$$

The volume of the sphere is $52.49kg/m^3$. Since the box has equal side length and must have the same volume as the sphere, then it is possible to obtain the side length by taking the third square root.

$$l_{box} = \sqrt[3]{52.49m^3} = 3.74m \tag{6.8}$$

Now that we know the side length of this first order approximation of the box, we can construct and illustrate it. We know that the center of the box is placed in $c =< x = 0, y = 0, z = -4.504 >$.

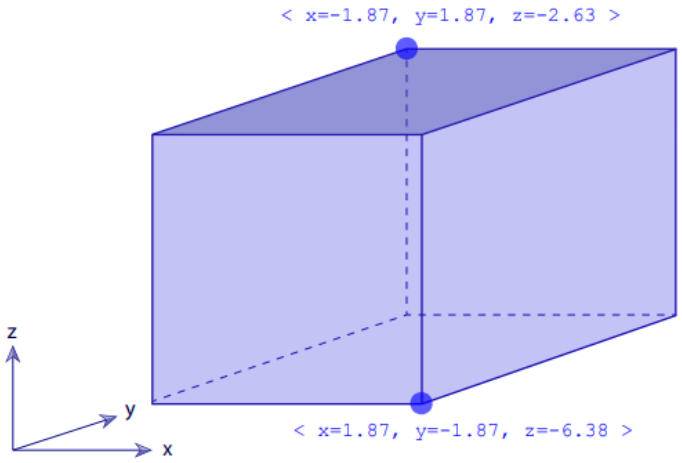

Figure 6.3: This figure shows the first order approximation to the true box

Now the first order approximation to the true unknown box is found. However, it is of interest to estimate the standard deviation of the model parameters between the true unknown box, and the first order approximation. This can be done by using the same procedure once again and construct a second order approximation and assuming that the standard deviation between the true model parameters and the first order approximation model parameters are the same as the standard deviation between the first order approximation of the model parameters and the second order approximation of the model parameters. The exact same technique is used, and the second order approximation is shown in figure 6.4

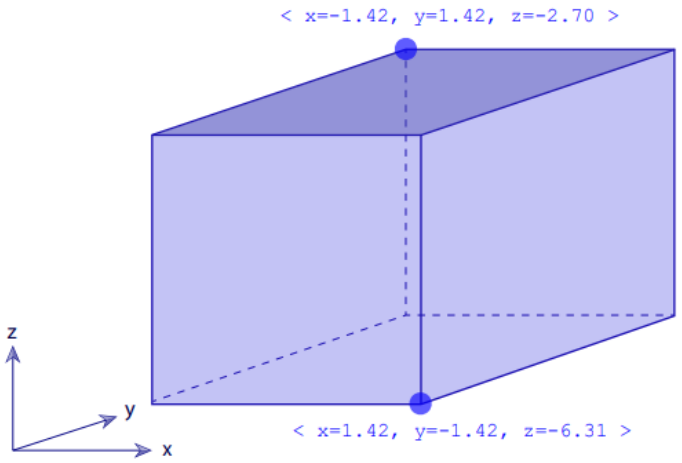

Figure 6.4: This figure shows the second order approximation to the true box

As mentioned before, the second order approximation is used to find the standard deviation between the true unknown model parameters and the first order approxima-

tion. This standard deviation will be used for the sampling of the informed proposal algorithm since the informed proposal algorithm is using a perturbation where the random generated number is multiplied with 3 times the standard deviation. Since the boxes are of equal side length, each model parameter will have the same standard deviation of $\sigma_{std} = 0.23m$. Since everything needed for the informed proposal sampling is known, we can construct the perturbation. Each perturbation step is calculated as the model parameter corresponding to the first order approximation multiplied with 3 times the standard deviation. Furthermore, the acceptance probability is changed. This is done since the true posterior $\sigma(\mathbf{m})$ and the approximate posterior (the informed proposal) need a probability function $q(\mathbf{m}_{n+1}|\mathbf{m}_n)$ to secure that the sampling is done without bias. The acceptance probability is now given by

$$p_{acc} = \frac{\sigma(\mathbf{m}_{n+1})}{\sigma(\mathbf{m}_n)} \frac{q(\mathbf{m}_n)}{q(\mathbf{m}_{n+1})} \tag{6.9}$$

where n+1 denotes the perturbed model and n denotes the current model. Once again, the logarithm of each function is used to secure numerical stability. This leads to the following rewriting of equation 6.9

$$\frac{\sigma(\mathbf{m}_{n+1})}{\sigma(\mathbf{m}_n)} \frac{q(\mathbf{m}_n)}{q(\mathbf{m}_{n+1})} = e^{log(\sigma(\mathbf{m}_{n+1})) - log(\sigma(\mathbf{m}_n)) + log(q(\mathbf{m}_n)) - log(q(\mathbf{m}_{n+1}))} \tag{6.10}$$

$q(\mathbf{m}_{n+1}|\mathbf{m}_n)$ is given in the same way the prior is given according to the implementation section.

$$q\left(\mathbf{m}\right) = k \ \exp\left\{\left(-\frac{1}{2}\left(\mathbf{m}-\mathbf{m}_0\right)^T \mathbf{C}_m^{-1}\left(\mathbf{m}-\mathbf{m}_0\right)\right)\right\}, \tag{6.11}$$

$\mathbf{C}_m$ is the co-variance matrix, with the standard deviation squared in the diagonal. The Pyhon code implementation of the IPMC for this box experiment is shown in the appendix section 12.2.

6.2 Box Experiment with IPMC

We remember that the goal of using the informed proposal MCMC algorithm is to test if the algorithm will improve the performance compared to the MCMC algorithm. The improvement can be shown by plotting the burn-in phase for both algorithms over a fixed numbers of iterations. By looking at the first order perturbation, we see that it has starting values close to the true unknown values. By taking that into account, we can predict that the IPMC model will do well in order to find acceptable model parameters from the sampling. Therefore, it is expected that the burn-in phase will be shorter for the IPMC algorithm compared to the MCMC algorithm. However, that would not be known prior to the sampling in the real world experiment. By looking at figure 6.5, the two samplings are shown over 40,000 iterations, for both the

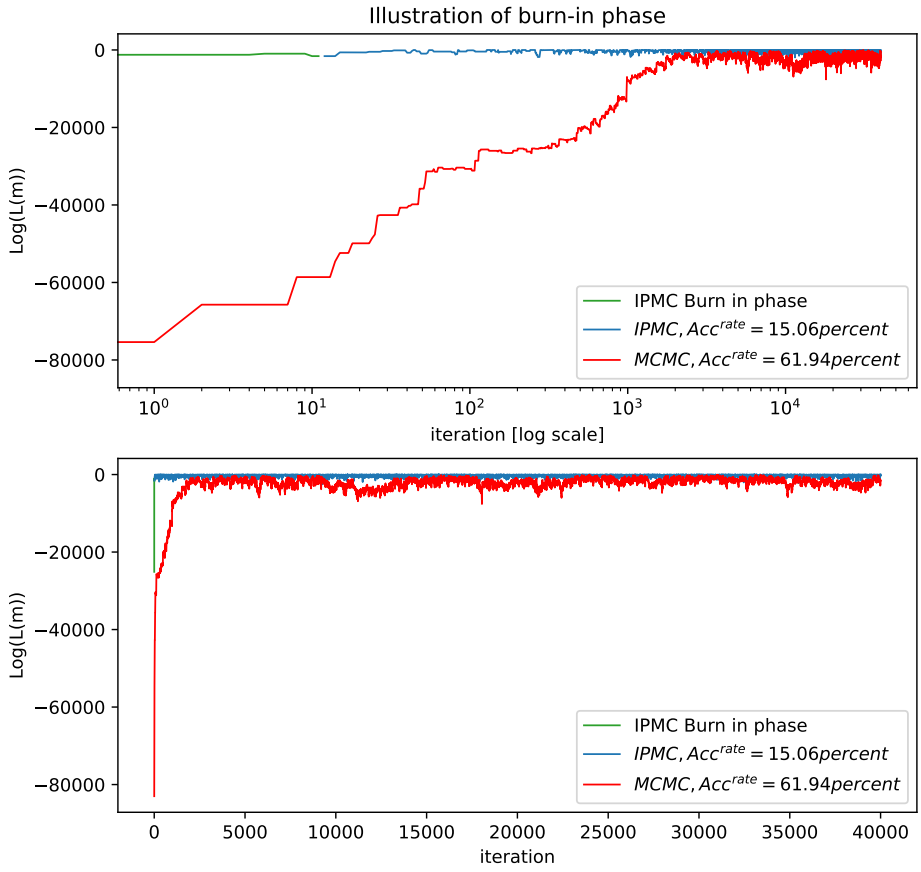

Figure 6.5: Both curves show convergence towards equilibrium of our Informed Proposal Monte Carlo (IPMC) method and a Markov Chain Monte Carlo (MCMC) method. The IPMC algorithm was guided by the approximations. The blue curve shows the convergence of a simple IPMC algorithm, with the green indicading the burn in phase and the red curve shows the simple MCMC with a Gaussian proposal perturbing one parameter at a time, and tuned to an acceptance rate of around 61 percent. The IPMC sampling resulted in a acceptance rate of around 15 percent. The IPMC sampling turned out to reach equilibrium 200 times faster than the MCMC sampling

MCMC algorithm and the IPMC algorithm. It is evident that the burn-in phase of the IPMC is much shorter than for the MCMC algorithm. A shorter period for the burn-in phase at around 200-300 times less iteration is evident, which is a reduction of around 2,000 to 3,000 iterations. By looking at 6.6, the sampling distributions of all 6 model parameters is shown. The number above each distribution denotes the highest bar found; marked in orange. The highest bar is the model parameter sampled that is classified as the most occurring, and it is the most accepted estimate of the model parameter. By looking at the values, it is evident that the sampling

finds an decent estimation of the model parameter at the value that occurs most frequently. Furthermore, this is evident when comparing with figure 6.1. Additionally, it is relevant to study how the IPMC algorithm behaves. A few tests were made to study if the IPMC algorithm performed better if the perturbation "size" increased or decreased. This means that the number which the first order approximation is multiplied with is increased. It is evident that by increasing the value of the number from 3 to a higher number, the number of accepted models is increased.

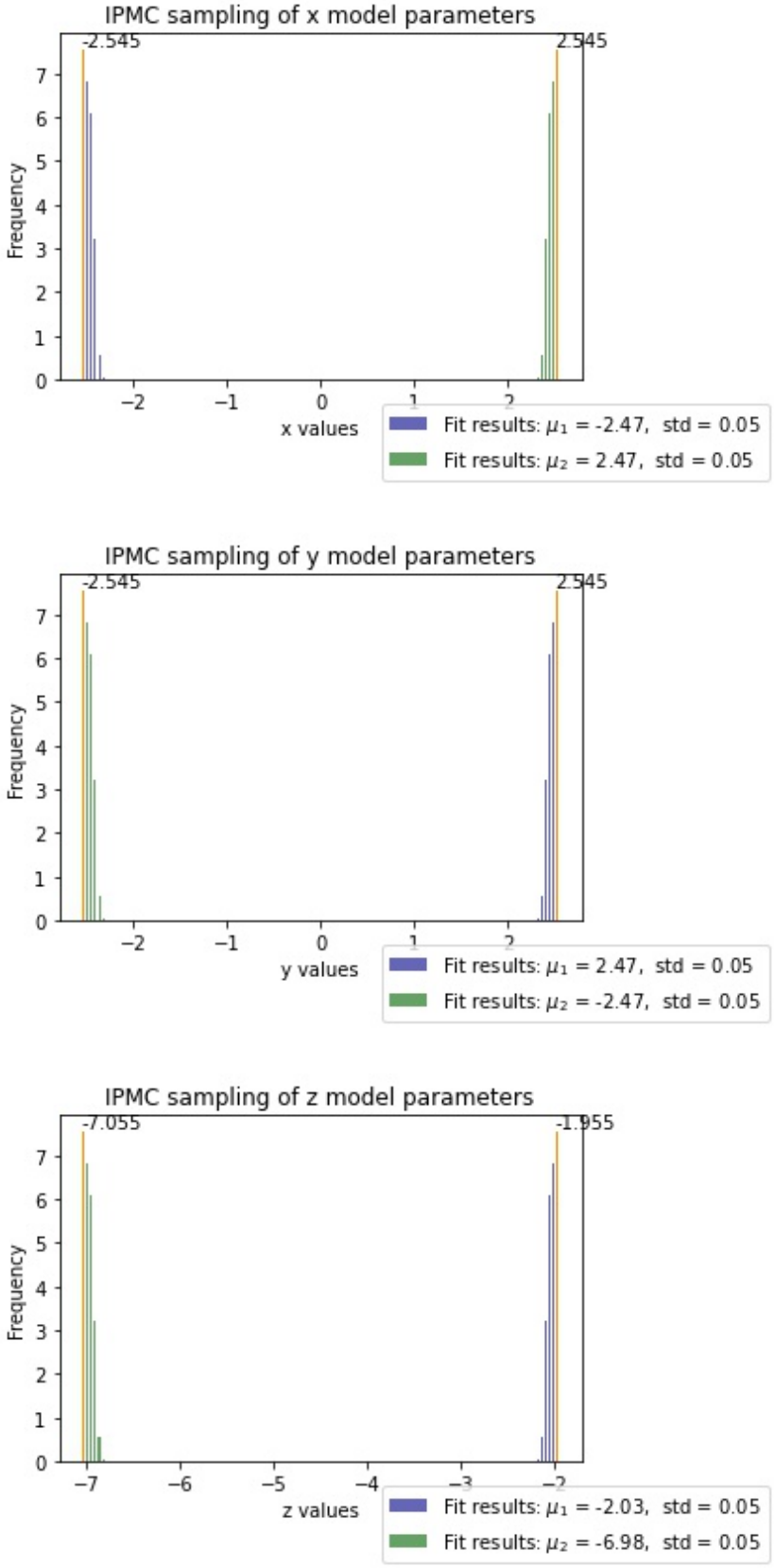

Figure 6.6: This figure shows how the IPMC sampling distributes the sampling for each of the six model parameters. The true values are shown in figure 6.1, and the highest bar in the histograms is denoted above. The statistical analysis of the probability density is presented in the legend outside each plot.

7. Acoustic Wave Experiment

Now it is desired to use the MCMC method and the IPMC method on a more advanced physical problem. The principle of the box experiment will be extended, however, the two types of algorithm IPMC and MCMC are once again used to sample the desired model parameters. The more advanced problem consists of an acoustic wave. The idea is to see if a highly non-linear inverse problem can be solved by a MCMC algorithm and improved by an IPMC algorithm. Keep in mind that a few mathematical steps can provide a lot of information to the algorithm which extends it to a informed algorithm. More specifically, the MCMC is improved by using approximations of the real problem to sample the true model parameters. To explain the new, more advanced problem, a wall constructed of different layers is considered. The thickness of the different layers is desired to be estimated since the thickness of the layers is the model parameters.

Imagine a building restoration project where the thickness of the layers of a wall can

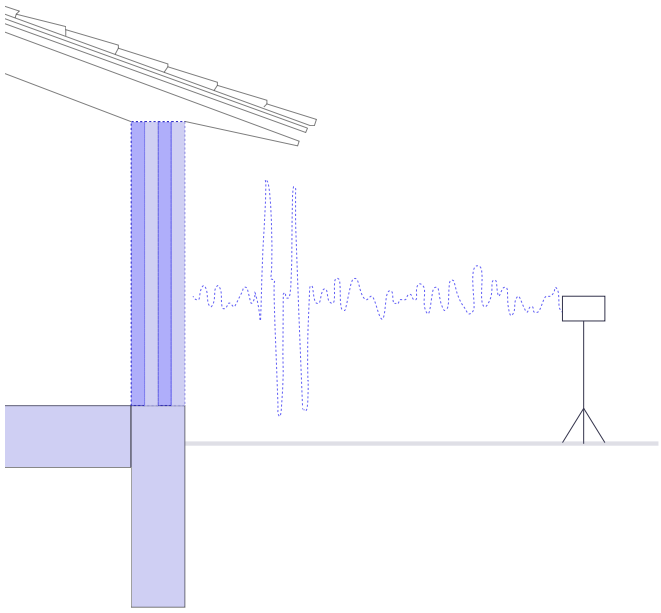

Figure 7.1: Principle of the acoustic wave experiment with 4 model parameters. The wall consists of two types alternating materials where the thickness is the desired model parameters. Adobe Illustrator is used to generate this figure.

be estimated by using an instrument sending an acoustic wave into the wall. The wall contains a reflectivity constant, while it is remembered that the reflecting wave

is used as observed data. The acoustic reflectivity constant between the alternating materials is therefore known. Since the main focus is on the optimization of algorithm iterations, only the needed wave theory is explained for this example.

7.1 Implementation: Forward function and Modelization error

Figure 7.2 shows the principle of a cosine Gauss Wavelet used for this experiment. The shown wavelet is used to create the intended forward function and to generate the observed data for this experiment. Equation 7.1 is used to construct the wavelet wWhere N is the number of samples of the wavelet, and T is the dominant period. Note that $N/2+1:N/2$ means that the wavelet data is generated in from N half plus one to N half.

$$\psi = cos(\frac{N}{2}+1:\frac{N}{2}\cdot\frac{2\cdot\pi}{T})\cdot exp(\frac{1}{2}\cdot(\frac{N}{2}+1:\frac{N}{2})^2\cdot\frac{1}{2\cdot N} \tag{7.1}$$

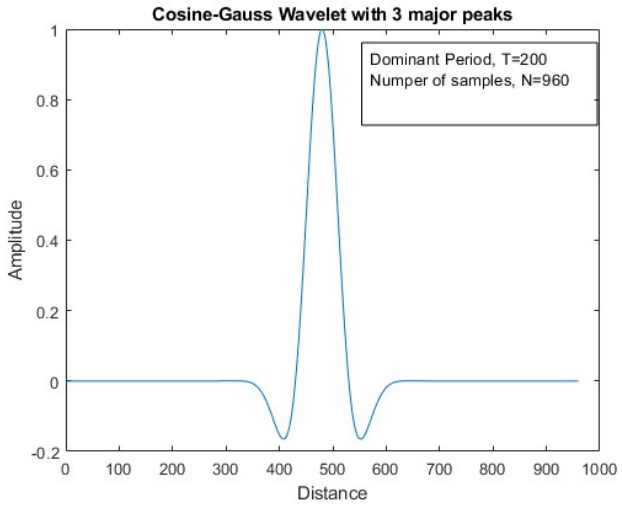

Figure 7.2: This figure shows the principle of a a cosine Gauss Wavelet. 3 major peaks are considered.

The wavelet is used to generate the observed data. This leads to a convolution between the array containing the reflectivity constant and the wavelet. This is done to create an array containing the desired waves inside the desired interval. The interval used is between 1 and 2,500 in a unit-less scale. The likelihood and the prior is constructed according to equation 3.3 and equation 3.2 in section 3.2. Once the observed data are calculated, the sampling simulation is ready to be done. During the simulation, the residuals are calculated as the difference between the calculated waves and the observed data. This is done to find the misfit summed with a prior.

We remember that this is done to find the acceptance probability in every iteration during the simulation. Once again the simulation is done over a large number of iterations. The choice of iterations is based on the burn in phase which means that it is adjusted due to the "size" of the burn in phase. An iteration number of 90,000 was chosen to get a clear view of the burn in phase. Furthermore, the IPMC was constructed since the goal is to compare the burn-in phase for the IPMC and the MCMC algorithms. Similar to the box experiment, once again the first and the second order approximation to the forward function was done. This time the first order approximation is made from a de-convolution of the model parameters, and the signal is the observed data. A de-convolution means that a wavelet is removed from a signal, which is the opposite of making a convolution. In the process of making a de-convolution, a matrix $\mathbf{G}$ is constructed from the in-build function called toeplitz which creates a type of matrix with constant values in the diagonal. Furthermore, Tikonov is used to generate the acoustic wave approximation:

$$\mathbf{ref}_{approx} = (\mathbf{G}^T\mathbf{G} + \epsilon * \mathbf{I})^{-1}(\mathbf{G}^T\mathbf{d}_{obs}) \tag{7.2}$$

where $\mathbf{ref}_{approx}$ is the desired approximation, and $\mathbf{I}$ is the identity matrix corresponding to the size of the observed data. For the specific approach here, Matlab code 12.5 in the appendix is used.
That leads to the first figure in figure 7.3. The peaks correspond to the first order approximation, and the first order approximation is now used for the IPMC simulation. Since the first order approximation does not provide information on the standard deviation between the true model and the first order approximation. Furthermore, another de-convolution is provided, and it is assumed that the standard deviation between the first order approximation and the second order approximation is the same as the standard deviation between the true model and the first order approximation. We remember that we obtain the modelization error by calculating the standard deviation between first and second order approximation. This leads to an implementation where the "true-and unknown" model parameters is sampled from the posterior probability density function. Additionally, the prior is used, and the function q is used to calculate the acceptance probability to ensure no bias during the sampling.

7.2 Acustic Wave Experiment with 4 and 20 Model parameters

The first and the second order approximations were found for the experiment with 4 and 20 model parameters. However, only the read of graphs for the experiment with

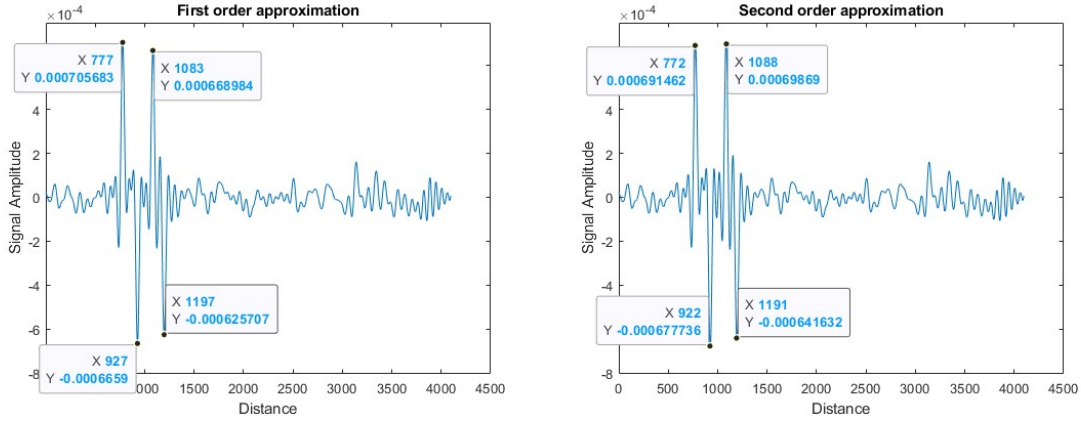

Figure 7.3: First and second order approximation for the acoustic experiment. A deconvolution method is used to make the approximations to obtain the desired modelization error. The values for both approximations are found by reading the values of the top of the highest peaks, and the standard deviation between two corresponding peaks is used for the IPMC sampling.

4 model parameters is presented. By looking at the two approximations, it is evident that the first and the second order approximations are close to each other, however, they differ by around a factor of five. This leads to a low standard deviation during the sampling and narrow sampling distributions. We remember that a low standard deviation between the two approximation results in a higher guidance of the informed proposal algorithm. In each perturbation, the first order approximation is multiplied with 3 times the standard deviation, and the function q contains the covariance matrix with 3 times the variance in the diagonal. This is done to get a larger area in which the model parameters are sampled. Furthermore, the noise affects the area in which the model parameters are sampled. The python code implementation of the MCMC and the IPMC for the acoustic wave experiment is shown in appendix section 12.3 and 12.4.

By looking at figure 7.4, it is evident that the four model parameters were sampled, and the sampling distributions are distributed fairly well around the true model parameters. The acceptance rate of the sampling is within the intended interval of 30-70 percent. The mean of the sampling for each model parameter is presented in the box next to the figure and the bar with most sample values has its value denoted over the bar since both the mean of the sampling and the highest bar are relevant in order to investigate if the sampling was done correctly. By looking at the burn-in phase shown in figure 7.5, once again the log likelihood function is plotted. Compared to earlier, the MCMC sampling was modified slightly. The noise, the step size and the number of iteration were changed compared to 5.4. This was changed to obtain a better comparison with the IPMC. It is evident that the IPMC algorithm was much faster than the MCMC algorithm. To get the best comparison, both algorithms were

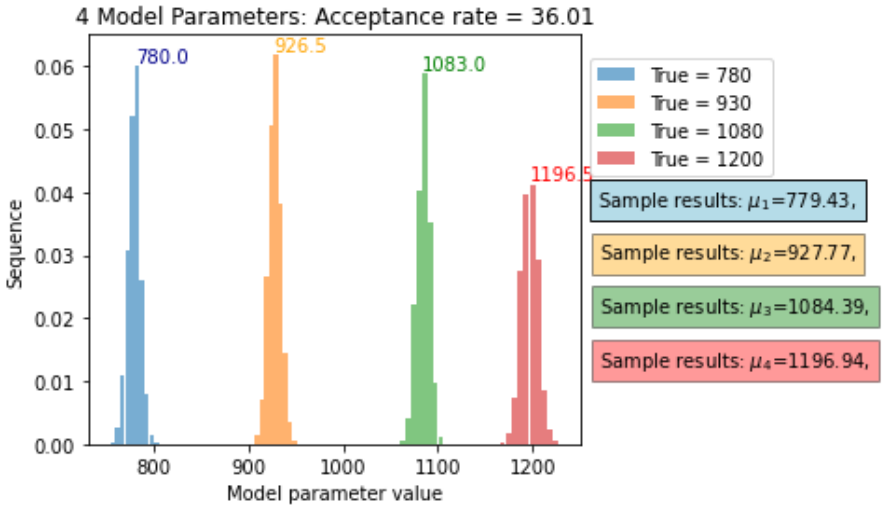

Figure 7.4: This figure shows how the IPMC sampling distributes the sampling for each of the four model parameters. The true values are shown and the highest pointing bar is noted with the values above. The mean of each distribution is shown in the four boxes outside the plot.

starter in the limit of the experiment. This means all four model parameters had an initial value of 2,500 since the experiment is defined in the interval from 1 to 2,500, in a unit-less scale. The informed proposal uses around 7-10 iterations to real the equilibrium close to zero, however, the MCMC algorithm uses around 10,000 iterations to reach the same equilibrium close to zero. The number of iterations is therefore reduced by a thousand times for the IPMC algorithm compared to the MCMC.

Furthermore, the sampling was done with 20 model parameters. By looking at figure 7.6, once again the log likelihood is plotted to investigate the number of iteration to obtain the desired equilibrium close to zero. Once again, the improvement is extensive for the IPMC compared to the MCMC. The IPMC algorithm converges at around 20 iterations and the MCMC converges at around 30,000 iterations. That is an improvement of around 1,500 times less iterations for the IPMC algorithm.

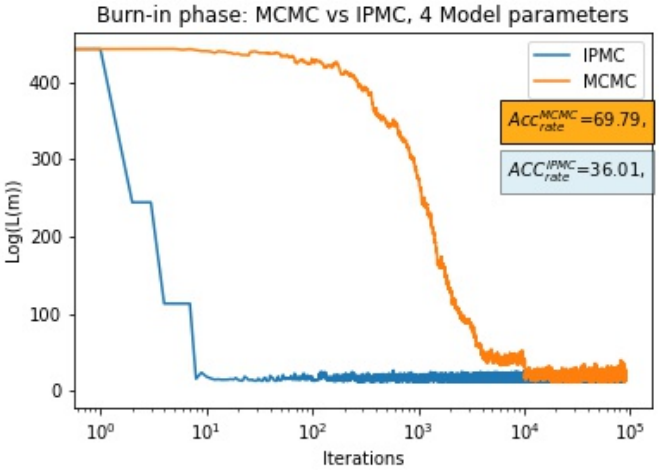

Figure 7.5: Both curves show convergence towards equilibrium. An Informed Proposal Monte Carlo (IPMC) method and a Markov Chain Monte Carlo (MCMC) method are used. The IPMC algorithm was guided by the approximations of making a de-convolution. The blue curve shows the convergence of a simple IPMC algorithm, and the orange curve shows the simple MCMC with a Gaussian proposal perturbing one parameter at a time and tuned to an acceptance rate below 70 percent. The IPMC sampling resulted in a acceptance rate of around 36 percent. The IPMC sampling turned out to reach equilibrium 1,000 times faster than the MCMC sampling.

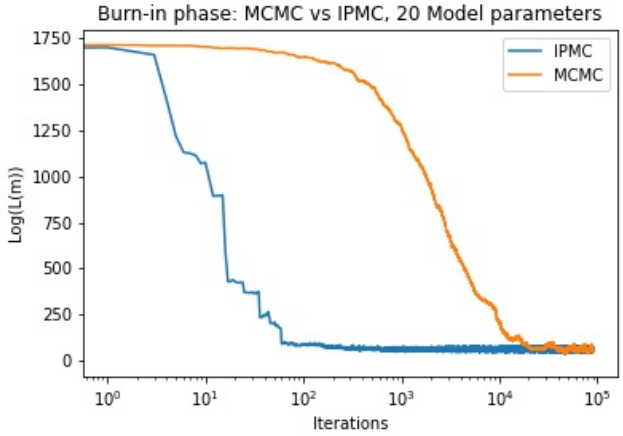

Figure 7.6: Both curves show convergence towards equilibrium. An Informed Proposal Monte Carlo (IPMC) method and a Markov Chain Monte Carlo (MCMC) method are used with 20 model parameters. The IPMC algorithm was guided by the approximations of making a deconvolution. The blue curve shows the convergence of a simple IPMC algorithm, and the orange curve shows the simple MCMC with a Gaussian proposal perturbing one parameter at a time.

8. Discussion

The Monte Carlo method is a strong choice of sampling algorithm since it enables us to sample probabilistic problems. However, one of the challenges associated with the use of Monte Carlo sampling is due to the highly non-linear and high-dimensional problems which slow down the efficiency for the MCMC algorithm, and the sampling is often more iteration consuming than first thought.

In this thesis, the main focus is based on two experiments. In the first experiment, the gravity field over a box was considered to estimate the model parameters (corner points) of the box from uncertainty analysis of the posterior. The second experiment was the an acoustic wave experiment based on a wavelet used to estimate the model parameters inside a materiel for example a wall. For both experiments, a clear improvement was visible in the burn-in phase when a MCMC and an IPMC were compared. For all experiments, the model parameters were estimated and showed excellent and precise results.
We remember the main idea, when going from a MCMC to an IPMC algorithm, is that the proposal of the algorithm is the key. The physical proposal is not resulting from the prior constraints of the model parameters (Khoshkholgh *et al.*, 2021b). In general, due to the q in equation 4.8 and figure 4.1, the prior will assign a different probability for a different solution. For a proposal it is different. We know that a proposal only has an impact on the frequency of the presented model to the acceptance and rejection criteria. Furthermore, when q (equation 4.8) is added to the acceptance probability the bias of the proposal will be counterbalanced asymptotically. This is due to a recompense in the acceptance probability. This means when a model with high probability is proposed, the acceptance probability is decreased. This was evident in the results of figure 7.5 and figure 6.5 since a lower acceptance rate was evident for the IPMC compared to a higher acceptance rate in the MCMC.

It is important to remember that the acceptance rate is allowed to be below 30 percent for the IPMC algorithm. A rejection of a model is not time-consuming for the program. However, this can be adjusted by changing the factor that decides the area sampled away from the first order approximation. For the analysis in this project, the sampling was done with 3 times the standard deviation between first

and second order approximations. By looking at figure 6.6, it was, however, evident that 3 times the standard deviation was a fairly low choice since only half of the expected distribution seems to be visible. In figure 11.1 in the appendix, the results of a less guided IPMC is shown. In figure 7.4, 3 times the standard deviation seemed reasonable. We remember that multiplying the standard deviation with 3 makes the algorithm less guided than multiplying with 1. However, the multiplication with 3 seemed necessary.

Furthermore, to construct an IPMC, we use physics to build a posterior-like proposal, which we use as the first order approximation to the true problem. In that way, we obtain information on the true problem without knowing the exact solution and without starting the sampling of the problem. For both experiments, The box and acoustic wave, the first order approximations turned out to be close to the "true-unknown" models. However, the starting point for the IPMC and for the MCMC was the same, which resulted in obtaining a more clear view of the performance of the two algorithms when comparing them. It is evident since both burn in phases starts at the same $log(L(\mathbf{m}))$ value shown in figure 6.5, figure 7.5 and figure 7.6.
Additionally, in the process of implementing the methodology described earlier, we add external information to an existing blind method. It is important to remember that, we in the process of adding external information to the sampler, we create a purely specialized algorithm which can only be used for solving a specific problem according to the external information gained. However, the possibilities to add external information to a sampler are open if the relevant approximations can be constructed.

In general, the MCMC and the IPMC were constructed with the goal of getting an acceptance rate between 30 percent and 70 percent. However, the IPMC often showed a lower acceptance rate which was reasonable due the the low cost of a rejection. The MCMC algorithem could be tuned to have an acceptance rate within the desired interval by changing the step size, noise, and number of iterations. The noise was chosen by trial and error, and a noise around 15 times less than the maxima of the gravity peak was chosen. The most important parameter was the step size since decreasing the step size would increase the acceptance rate. Finally, the number of iterations needed to be fairly large, otherwise the MCMC sampler would not find the target distribution.

Since the IPMC algorithm is not using a step size, the story for tuning the acceptance rate is different. However, once again the noise has an influence. The noise was chosen to be the same as for the MCMC to get the best comparison. The space used for the sampling was adjusted to get decent samplings. If the space for the sampling was chosen to be too narrow, the algorithm would converge extremely slow

towards the target distribution, and the acceptance rate would be extremely small. This was slightly evident in figure 6.6 since a fairly small modelization error was found. However, the results showed the desired model parameters agreeable to the expectations. Finally, it was evident that both approximations according to both experiments were acceptable in order to construct the informed proposal algorithm. However, poor approximation cannot mislead the the algorithm asymptotically, but only affect the computational time (Khoshkholgh *et al.*, 2021a).

9. Conclusion

In this thesis, the Markov Chain Monte Carlo algorithm has been implemented and compared to an implemented Informed proposal Monte Carlo algorithm. Two examples have been investigated to make the analysis; the gravity box experiment and the acoustic wave experiment. For both experiments, the two types of algorithms were implemented successfully, and the model parameters were estimated extremely accurately. As expected, the burn-in phase of the IPMC was shorter than the burn-in phase of the MCMC algorithm.

For the box experiment, an improvement of around 200 times less iterations was obtained by implementing the IPMC algorithm. For the acoustic wave experiment, an improvement of around 1,000 iterations was obtained for 4 model parameters and an improvement of around 1,500 iterations was obtained for 20 model parameters. These improvements are excellent results, and the contribution, which an IPMC algorithm can provide to solve inverse problems is promising.

Furthermore, it is remembered that the performance of the IPMC algorithm depends on the constructed external proposal which is problem dependent. The aim is to construct similarity between the target distribution and the proposal since this will result in higher sampling efficiency. This results in the conclusion that this methodology can be implemented to any type of MCMC sampling algorithm if a proposal probability density can be constructed as an approximation to the posterior.

10. Bibliography

Bongiolo, Alessandra, Jeferson Souza, Francisco Ferreira, and Luís Castro (Sept. 2013). „A MATLAB®/octave program to generate gravity and magnetic anomalies due to rectangular prismatic bodies“. In: *Revista Brasileira de Geofisica* 31, pp. 347–363.

Khoshkholgh, Sarouyeh, Andrea Zunino, and Klaus Mosegaard (2021a). „Full-waveform Inversion by Informed-Proposal Monte Carlo“. In: *Earth and Space Science Open Archive*, p. 17.

Khoshkholgh, Sarouyeh, Andrea Zunino, and Klaus Mosegaard (Apr. 2021b). „Informed proposal Monte Carlo“. In: *Geophysical Journal International* 226.2, pp. 1239–1248. eprint: `https://academic.oup.com/gji/article-pdf/226/2/1239/38017494/ggab173.pdf`.

Mosegaard, Klaus (2006). „Monte Carlo analysis of inverse problems“. English. PhD thesis.

Mosegaard, Klaus (2012). „Limits to Nonlinear Inversion“. English. In: *Applied Parallel and Scientific Computing. Proceedings of the PARA 2010 Meeting*. Lecture Notes in Computer Science Part 1. 10th International Conference on Applied Parallel and Scientific Computing : PARA 2010 ; Conference date: 06-06-2010 Through 09-06-2010. Springer, pp. 11–21.

Mosegaard, Klaus and Malcolm Sambridge (Apr. 2002). „Monte Carlo analysis of inverse problems“. In: *Inverse Problems* 18.3, R29–R54.

Mosegaard, Klaus and Albert Tarantola (1995). „Monte Carlo sampling of solutions to inverse problems“. In: *Journal of Geophysical Research: Solid Earth* 100.B7, pp. 12431–12447. eprint: `https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/94JB03097`.

Sambridge, Malcolm and Klaus Mosegaard (2002). „MONTE CARLO METHODS IN GEOPHYSICAL INVERSE PROBLEMS“. In: *Reviews of Geophysics* 40.3, pp. 3-1-3–29. eprint: `https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2000RG000089`.

Tarantola Albert Valette, B. (1982). „Inverse problems = Quest for information“. In: *Journal of Geophysics*.

11. Appendix

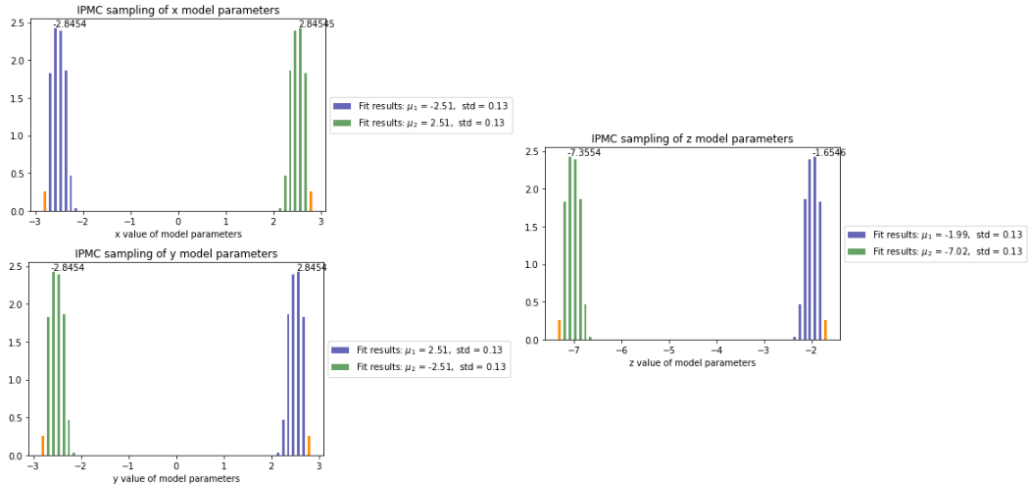

Figure 11.1: Shows the sampling with a less guided informed proposal

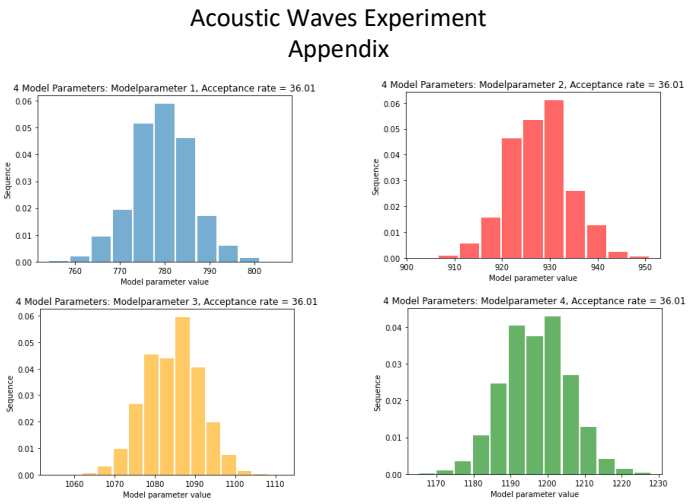

Figure 11.2: In addition to figure 7.4, the model parameters are plotted individually

12. Appendix Code

12.1 Markov Chain Monte Carlo Box

```
import numpy as np
import matplotlib.pyplot as plt
import numba as nb
from numba import jit
from tqdm.notebook import tqdm, trange
from matplotlib.colors import Normalize
from scipy.stats import norm
from scipy.stats import norm
from numba import njit, objmode
from mpl_toolkits import mplot3d
d_obs = np.loadtxt("msdata.txt", delimiter = ",") #mgal / cm/s^2
x_sensors = np.loadtxt("xmsdata.txt", delimiter = ",")
y_sensors = np.loadtxt("ymsdata.txt", delimiter = ",")
n_obs = len(d_obs)
Nx = 2
er = 10**-(50)
noise = np.abs((np.mean(d_obs))/35)#  np.abs((np.mean(d_obs))/50) #mgal
d_obs
d_obs.shape
def betta1(i,b,y):
    betta = np.zeros((1,len(y)))
    betta=b[i]-y
    return betta

def alfa1(i,m,x):
    alfa = np.zeros((1,len(x)))
    alfa=m[i]-x
    return alfa

def R1(i,j,k,m,b,x,y,h): # Beregner værdierne for de afstande, der bruges i "F" -f
```

```
    R = np.zeros((1,len(y)))
    R=np.array([np.sqrt(alfa1(i,m,x)**2+betta1(j,b,y)**2+h[k]**2)])
    return R


def f1(m,b,h,x,y,i,j,k,s): # Funktion "f" for formlen ifølge Plouff (1976).
    f = np.zeros((1,len(y)))
    f=s[i]*s[j]*s[k]*((h[k]*np.arctan(alfa1(i,m,x)*betta1(j,b,y)/(er+R1(i,j,k,m,b,
        +betta1(j,b,y))-betta1(j,b,y)*np.log(R1(i,j,k,m,b,x,y,h)+alfa1(i,m,x))))
    return f

def g(m,b,h): #mgal
    #m = np.array([-2.54, 2.54])
    #b = np.array([2.54, -2.54])
    #h = np.array([-2.0,-4.0]) #np.array([-2, -7.0800])
    qx = np.linspace(-10,10,100)
    qy = np.linspace(-10,10,100)
    x, y = np.meshgrid(qx, qy, sparse=False, indexing='ij')
    s = np.array((1,len(np.array([-2.0,-4.0]))))
    s[0]=-1
    s[1]=1
    ac=(m[1]+m[0])/2 # Finder koordinaterne for prismaets centrum og flytter syste
    bc=(b[1]+b[0])/2 # koordinater for dette center.
    x=x-ac
    y=y-bc
    m=m-ac
    b=b-bc


    g = np.zeros((1,len(y)))
    g=6.67*10**(-11)*(f1(m,b,h,x,y,0,0,0,s)+f1(m,b,h,x,y,0,0,1,s)+f1(m,b,h,x,y,0,1
            f1(m,b,h,x,y,1,0,0,s)+f1(m,b,h,x,y,1,0,1,s)+f1(m,b,h,x,y,1,1,0,s)+f1(m

    return g[0,:]


g(np.array([-2.54, 2.54]),np.array([2.54, -2.54]),np.array([-2.0,-4.0])).shape
#constructing the log likelighood function
Cd = np.eye(len(d_obs))*(noise)**2
```

```
Cd_inv = np.linalg.inv(Cd)
def log_L(m,b,h):
    mitfit_m = d_obs - g(m,b,h) #misfit
    return np.min(-0.5 * (mitfit_m.T@Cd_inv@mitfit_m)) #Cd_inv, Cd er std noise
log_L(np.array([-2.54, 2.54]),np.array([2.54, -2.54]),np.array([-2.0,-4.0])).shape
# run MCMC iteration
iterations = 40000
i_creation = iterations
j_creation = iterations
log_L_mat = np.zeros((iterations - 1))
m = np.zeros((iterations, Nx))+np.array([-2,2])#simulate Nx model paraters [-7,7],
b = np.zeros((iterations, Nx))+np.array([2,-2])
h = np.zeros((iterations, Nx))+np.array([-1,-3])

step = 0.9  #0.004                #decreasing step increasing acc rate
accepted_model = np.zeros(iterations - 1) #create the zero base for counting accep
accepted_model_b = np.zeros(iterations - 1)
accepted_model_h = np.zeros(iterations - 1)


#log_L_mat =
#log_L_bat =
#log_L_hat =

for i in trange(iterations - 1):
    m_curr = m[i, :].copy() #to secure current model has same dimensions as initia
    b_curr = b[i, :].copy()
    h_curr = h[i, :].copy()

    log_L_mat[i] = log_L(m_curr,b_curr,h_curr) #calculate lokelihood of the curren


    rnd_index = np.random.randint(len(m_curr)) #create a random index
    rnd_index_b = np.random.randint(len(b_curr)) #create a random index
    rnd_index_h = np.random.randint(len(h_curr)) #create a random index

    m_pert = m_curr.copy() #copy current model
    b_pert = b_curr.copy() #copy current model
    h_pert = h_curr.copy() #copy current model
```

```
u = np.random.rand()
rnd_pert = np.random.randint(1,5)
if rnd_pert ==1:
    m_pert[rnd_index] += (2. * u - 1.) * step #perturb current model
if rnd_pert ==2:
    b_pert[rnd_index_b] += (2. * u - 1.) * step #perturb current model
else:
    h_pert[rnd_index_b] += (2. * u - 1.) * step #perturb current model


log_pL_curr = log_L(m_curr,b_curr,h_curr) #calculate likelihood of current mod
log_pL_pert = log_L(m_pert,b_pert,h_pert) #calculate likelihood of peterbed mo


p_accept = min(1, np.exp(log_pL_pert - log_pL_curr).all()) #metropolis accepta


u = np.random.rand() #create new random value for pertubation
if u < p_accept:
    m[i + 1, :] = m_pert
    accepted_model[i] = 1
else:
    m[i + 1, :] = m_curr

if u < p_accept:
    b[i + 1, :] = b_pert
    accepted_model_b[i] = 1
else:
    b[i + 1, :] = b_curr
if u < p_accept:
    h[i + 1, :] = h_pert
    accepted_model_h[i] = 1
else:
    h[i + 1, :] = h_curr

#print("pacc {}".format(p_accept))
#print("log_pL_pert {}".format(log_pL_pert))
#print("log_pL_curr {}".format(log_pL_curr))
```

```
print("acceptance rate {}%".format(sum(accepted_model_b) /
                                   iterations * 100))
print("number of accepted models {}".format(sum(accepted_model_b)))
cutoff = 10000
m_wo_burn = m[cutoff:, :]
m_mean = np.average(m_wo_burn, axis=0)
m_std = np.std(m_wo_burn, axis=0)

b_wo_burn = b[cutoff:, :]
b_mean = np.average(b_wo_burn, axis=0)
b_std = np.std(b_wo_burn, axis=0)

h_wo_burn = h[cutoff:, :]
h_mean = np.average(h_wo_burn, axis=0)
h_std = np.std(h_wo_burn, axis=0)
# Plot the histogram.
mu1, std1 = norm.fit(m_wo_burn[:,0])
mu2, std2 = norm.fit(m_wo_burn[:,1])
plt.hist(m_wo_burn[:,0],label="Fit results: $\mu_{1}$ = %.2f,  std = %.2f" % (mu1,
plt.hist(m_wo_burn[:,1],label="Fit results: $\mu_{2}$ = %.2f,  std = %.2f" % (mu2,
plt.legend()

# Plot the histogram.
mu1, std1 = norm.fit(b_wo_burn[:,0])
mu2, std2 = norm.fit(b_wo_burn[:,1])
plt.hist(b_wo_burn[:,0],label="Fit results: $\mu_{3}$ = %.2f,  std = %.2f" % (mu1,
plt.hist(b_wo_burn[:,1],label="Fit results: $\mu_{4}$ = %.2f,  std = %.2f" % (mu2,
plt.legend()
# Plot the histogram.
mu1, std1 = norm.fit(h_wo_burn[:,0])
mu2, std2 = norm.fit(h_wo_burn[:,1])
plt.hist(h_wo_burn[:,0],label="Fit results: $\mu_{5}$ = %.2f,  std = %.2f" % (mu1,
plt.hist(h_wo_burn[:,1],label="Fit results: $\mu_{6}$ = %.2f,  std = %.2f" % (mu2,
plt.legend()
# Plot the histogram.
plt.figure(figsize=(8, 6))
mu1, std1 = norm.fit(m_wo_burn[:,0])
mu2, std2 = norm.fit(m_wo_burn[:,1])
plt.hist(m_wo_burn[:,0],label="Fit results: $\mu_{1}$ = %.2f,  std = %.2f" % (mu1,
```

```
plt.hist(m_wo_burn[:,1],label="Fit results: $\mu_{2}$ = %.2f,  std = %.2f" % (mu2,
plt.legend()
# Plot the histogram.
mu1, std1 = norm.fit(b_wo_burn[:,0])
mu2, std2 = norm.fit(b_wo_burn[:,1])
plt.hist(b_wo_burn[:,0],label="Fit results: $\mu_{3}$ = %.2f,  std = %.2f" % (mu1,
plt.hist(b_wo_burn[:,1],label="Fit results: $\mu_{4}$ = %.2f,  std = %.2f" % (mu2,
plt.legend()
# Plot the histogram.
mu1, std1 = norm.fit(h_wo_burn[:,0])
mu2, std2 = norm.fit(h_wo_burn[:,1])
plt.hist(h_wo_burn[:,0],label="Fit results: $\mu_{5}$ = %.2f,  std = %.2f" % (mu1,
plt.hist(h_wo_burn[:,1],label="Fit results: $\mu_{6}$ = %.2f,  std = %.2f" % (mu2,
plt.legend()
plt.title("Six model parameters sampled")
plt.xlabel("Model Parameter value")
plt.ylabel("Sequence")
plt.savefig('all_parameters2.jpg')
```

12.2 Informed proposal Monte Carlo Box

```
import numpy as np
import math
import matplotlib.pyplot as plt
import numba as nb
from numba import jit
from tqdm.notebook import tqdm, trange
from matplotlib.colors import Normalize
from scipy.stats import norm
from scipy.stats import norm
from numba import njit, objmode
from mpl_toolkits import mplot3d
import random
from matplotlib.offsetbox import AnchoredText
random.seed(142)
d_obs = np.loadtxt("msdata.txt", delimiter = ",") #mgal / cm/s^2
mcmc_data = np.loadtxt("mcmc40kiter.txt", delimiter = ",")
x_sensors = np.loadtxt("xmsdata.txt", delimiter = ",")
```

```
y_sensors = np.loadtxt("ymsdata.txt", delimiter = ",")
n_obs = len(d_obs)
Nx = 2
er = 10**(-50)
noise = ...
noise
def betta1(i,b,y):
    betta = np.zeros((1,len(y)))
    betta=b[i]-y
    return betta

def alfa1(i,m,x):
    alfa = np.zeros((1,len(x)))
    alfa=m[i]-x
    return alfa

def R1(i,j,k,m,b,x,y,h): # Beregner værdierne for de afstande, der bruges i "F" -f
    R = np.zeros((1,len(y)))
    R=np.array([np.sqrt(alfa1(i,m,x)**2+betta1(j,b,y)**2+h[k]**2)])
    return R


def f1(m,b,h,x,y,i,j,k,s): # Funktion "f" for formlen ifølge Plouff (1976).
    f = np.zeros((1,len(y)))
    f=s[i]*s[j]*s[k]*((h[k]*np.arctan(alfa1(i,m,x)*betta1(j,b,y)/(er+R1(i,j,k,m,b,
        +betta1(j,b,y))-betta1(j,b,y)*np.log(R1(i,j,k,m,b,x,y,h)+alfa1(i,m,x))))
    return f

def g(m,b,h): #mgal

    qx = np.linspace(-10,10,100)
    qy = np.linspace(-10,10,100)
    x, y = np.meshgrid(qx, qy, sparse=False, indexing='ij')
    s = np.array((1,len(np.array([-2.0,-4.0]))))
    s[0]=-1
    s[1]=1
    ac=(m[1]+m[0])/2 # Finder koordinaterne for prismaets centrum og flytter syste
    bc=(b[1]+b[0])/2 # koordinater for dette center.
    x=x-ac
```

```
    y=y-bc
    m=m-ac
    b=b-bc


    g = np.zeros((1,len(y)))
    g=6.67*10**(-11)*(f1(m,b,h,x,y,0,0,0,s)+f1(m,b,h,x,y,0,0,1,s)+f1(m,b,h,x,y,0,1
            f1(m,b,h,x,y,1,0,0,s)+f1(m,b,h,x,y,1,0,1,s)+f1(m,b,h,x,y,1,1,0,s)+f1(m

    return g[0,:]

m0_tmp = np.array([-1.87, 1.87])
b0_tmp = np.array([1.87, -1.87])
h0_tmp = np.array([-2.63, -6.38])

m0 = np.concatenate((m0_tmp, b0_tmp, h0_tmp), axis=0)
Cm = np.eye(len(m0))*3*np.std(np.array([-1.42, -1.87]))**2
Cm_inv = np.linalg.inv(Cm)

def log_q(m,b,h): #spørgmål om denne kan være rigtig
    res = np.concatenate((m, b, h), axis=0)
    dm = res - m0
    return (-0.5 * (dm.T @ Cm_inv @ dm))
#constructing the log likelighood function   ln(e**x) = x
Cd = np.eye(len(d_obs))*(noise)**2
Cd_inv = np.linalg.inv(Cd)

def log_L(m,b,h):
    mitfit_m = d_obs - g(m,b,h) #misfit
    return (((-0.5 * (mitfit_m.T@Cd_inv@mitfit_m)))) #Cd_inv, Cd er std noise
# run MCMC iteration
iterations = 40000


log_L_mat = np.zeros((iterations - 1))
m = np.zeros((iterations, Nx))
b = np.zeros((iterations, Nx))
h = np.zeros((iterations, Nx))
```

```
accepted_model_m1 = np.zeros(iterations - 1)
accepted_model_m2 = np.zeros(iterations - 1)
accepted_model_b1 = np.zeros(iterations - 1)
accepted_model_b2 = np.zeros(iterations - 1)
accepted_model_h1 = np.zeros(iterations - 1)
accepted_model_h2 = np.zeros(iterations - 1)

for i in trange(iterations - 1):
    m_curr = m[i, :].copy() #to secure current model has same dimensions as initia
    b_curr = b[i, :].copy()
    h_curr = h[i, :].copy()

    m_pert = m_curr.copy() #copy current model
    b_pert = b_curr.copy() #copy current model
    h_pert = h_curr.copy() #copy current model

    log_L_mat[i] = np.min(log_L(m_curr,b_curr,h_curr)) #save log_l values into log

    # u skal være +/- 2-3 spredninger bred og have middelværdi  i 0, samt være uni
    u_m = np.random.uniform(-1,1)



    #pertubation of each model parameter
    m_pert[0] = m0_tmp[0]+u_m*-5*np.std(np.array([-1.42, -1.87]))
    m_pert[1] = m0_tmp[1]+u_m*5*np.std(np.array([1.42, 1.87]))
    b_pert[0] = b0_tmp[0]+u_m*5*np.std(np.array([1.42, 1.87]))
    b_pert[1] = b0_tmp[1]+u_m*-5*np.std(np.array([-1.42, -1.87]))
    h_pert[0] = h0_tmp[0]+u_m*5*np.std(np.array([-3.08,-2.63]))
    h_pert[1] = h0_tmp[1]+u_m*-5*np.std(np.array([-5.93,-6.38]))

    log_q_curr = log_q(m_curr,b_curr,h_curr)
    log_q_pert = log_q(m_pert,b_pert,h_pert)
    log_pL_curr = log_L(m_curr,b_curr,h_curr) #calculate likelihood of current mod
    log_pL_pert = log_L(m_pert,b_pert,h_pert) #calculate likelihood of peterbed mo


    p_accept = min(1, (np.exp(log_pL_pert - log_pL_curr+log_q_curr - log_q_pert)).
```

```
u = random.uniform(0,1) #normal virker vist også
if u < p_accept:
    m[i + 1, 0] = m_pert[0] #for all model parameters in the first coulom; x1,
    accepted_model_m1[i] = 1
else:
    m[i + 1, 0] = m_curr[0]

if u < p_accept:
    b[i + 1, 0] = b_pert[0]
    accepted_model_b1[i] = 1
else:
    b[i + 1, 0] = b_curr[0]

if u < p_accept:
    h[i + 1, 0] = h_pert[0]
    accepted_model_h1[i] = 1
else:
    h[i + 1, 0] = h_curr[0]

if u < p_accept:
    m[i + 1, 1] = m_pert[1] #for all model parameters in the second coulom; x2
    accepted_model_m2[i] = 1
else:
    m[i + 1, 1] = m_curr[1]

if u < p_accept:
    b[i + 1, 1] = b_pert[1]
    accepted_model_b2[i] = 1
else:
    b[i + 1, 1] = b_curr[1]

if u < p_accept:
    h[i + 1, 1] = h_pert[1]
    accepted_model_h2[i] = 1
else:
    h[i + 1, 1] = h_curr[1]
```

```
    #print("p_acc {}".format(p_accept))


print("acceptance rate {}%".format(sum(accepted_model_b1) /
                                   iterations * 100))
print("number of accepted models {}".format(sum(accepted_model_b1)))
cutoff = 150
m_wo_burn = m[cutoff:, :]
m_mean = np.average(m_wo_burn, axis=0)
m_std = np.std(m_wo_burn, axis=0)

b_wo_burn = b[cutoff:, :]
b_mean = np.average(b_wo_burn, axis=0)
b_std = np.std(b_wo_burn, axis=0)

h_wo_burn = h[cutoff:, :]
h_mean = np.average(h_wo_burn, axis=0)
h_std = np.std(h_wo_burn, axis=0)
# Plot the histogram.
mu1, std1 = norm.fit(m_wo_burn[:,0])
mu2, std2 = norm.fit(m_wo_burn[:,1])
y, x, bars= plt.hist(m_wo_burn[:,0],label="Fit results: $\mu_{1}$ = %.2f,  std = %
y1, x1, bars1= plt.hist(m_wo_burn[:,1],label="Fit results: $\mu_{2}$ = %.2f,  std
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.title('IPMC sampling of x model parameters')
plt.xlabel('x value of model parameters')
# Compute the max value (plt.hist returns the x and y positions of the bars)
ymax = y.max()
idx = np.where(y == ymax)[0][0]
xval = x[idx]
xmin = np.round(x.min(), 4)#x.min()
# Annotate the highest value
plt.gca().text(xval, ymax, xmin, ha='left', va='bottom')
# Make one bin stand out
bars[0].set_fc('darkorange') # Set color
```

```
bars[0].set_alpha(1) # Set opaci
# Compute the max value (plt.hist returns the x and y positions of the bars)
ymax1 = y1.max()
idx = np.where(y1 == ymax1)[0][0]
xval = x1[idx]
xmax1 = np.round(x1.max(),5) #x1.max()
# Annotate the highest value
plt.gca().text(xval, ymax1, xmax1, ha='left', va='bottom')
# Make one bin stand out
bars1[-1].set_fc('darkorange') # Set color
bars1[-1].set_alpha(1) # Set opaci

# Plot the histogram.
mu1, std1 = norm.fit(b_wo_burn[:,0])
mu2, std2 = norm.fit(b_wo_burn[:,1])
y, x,bars= plt.hist(b_wo_burn[:,0],label="Fit results: $\mu_{1}$ = %.2f,  std = %.
y1, x1,bars1 = plt.hist(b_wo_burn[:,1],label="Fit results: $\mu_{2}$ = %.2f,  std
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.title('IPMC sampling of y model parameters')
plt.xlabel('y value of model parameters')
# Compute the max value (plt.hist returns the x and y positions of the bars)
ymax = y.max()
idx = np.where(y == ymax)[0][0]
xval = x[idx]
xmin = np.round(x.max(),4)
# Annotate the highest value
plt.gca().text(xval, ymax, xmin, ha='left', va='bottom')
# Make one bin stand out
bars[-1].set_fc('darkorange') # Set color
bars[-1].set_alpha(1) # Set opaci
# Compute the max value (plt.hist returns the x and y positions of the bars)
ymax1 = y1.max()
idx = np.where(y1 == ymax1)[0][0]
xval = x1[idx]
xmax1 = np.round(x1.min(),4)
# Annotate the highest value
plt.gca().text(xval, ymax1, xmax1, ha='left', va='bottom')
# Make one bin stand out
bars1[0].set_fc('darkorange') # Set color
```

```
bars1[0].set_alpha(1) # Set opaci
# Plot the histogram.
mu1, std1 = norm.fit(h_wo_burn[:,0])
mu2, std2 = norm.fit(h_wo_burn[:,1])
y, x, bars = plt.hist(h_wo_burn[:,0],label="Fit results: $\mu_{1}$ = %.2f,  std =
y1, x1,bars1 = plt.hist(h_wo_burn[:,1],label="Fit results: $\mu_{2}$ = %.2f,  std
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.title('IPMC sampling of z model parameters')
plt.xlabel('z value of model parameters')
# Compute the max value (plt.hist returns the x and y positions of the bars)
ymax = y.max()
idx = np.where(y == ymax)[0][0]
xval = x[idx]
xmin = np.round(x.max(),4)
# Annotate the highest value
plt.gca().text(xval, ymax, xmin, ha='left', va='bottom')
# Make one bin stand out
bars[-1].set_fc('darkorange') # Set color
bars[-1].set_alpha(1) # Set opaci
# Compute the max value (plt.hist returns the x and y positions of the bars)
ymax1 = y1.max()
idx = np.where(y1 == ymax1)[0][0]
xval = x1[idx]
xmax1 = np.round(x1.min(),4)
# Annotate the highest value
plt.gca().text(xval, ymax1, xmax1, ha='left', va='bottom')
# Make one bin stand out
bars1[0].set_fc('darkorange') # Set color
bars1[0].set_alpha(1) # Set opaci
m_wo_burn = m[cutoff:, :]
log_L_burn = np.empty(cutoff)
log_L_states = np.empty(iterations - cutoff)

log_L_burn = log_L_mat[:cutoff]
log_L_states = log_L_mat[cutoff:]


#Saving the array in a text file
#np.savetxt("fem.txt",log_L_mat,delimiter=',');
```

```
iter_count = np.linspace(0, iterations, iterations - 1)

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(iter_count[:cutoff], log_L_burn, c="tab:green",linewidth=1.0,label="IPMC
plt.plot(iter_count[cutoff:], log_L_states, c="tab:blue",linewidth=1.0,label="$IPM
#plt.plot(iter_count, mcmc_data, c="red",linewidth=1.0,label="$MCMC$")
plt.xlabel("iteration")
plt.ylabel("Log(L(m))")
plt.legend()
plt.xscale("log")
plt.title("Illustration of burn-in phase [log scale]")

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(iter_count[:cutoff], log_L_burn, c="tab:green",linewidth=1.0,label="IPMC
plt.plot(iter_count[cutoff:], log_L_states, c="tab:blue",linewidth=1.0,label="$IPM
#plt.plot(iter_count, mcmc_data, c="red",linewidth=1.0,label="$MCMC$")


plt.xlabel("iteration")
plt.ylabel("Log(L(m))")
plt.legend()
plt.title("Illustration of burn-in phase")
```

12.3 Markov Chain Monte Carlo Acoustic Wave

```
import numpy as np
import matplotlib.pyplot as plt
import numba as nb
from numba import jit
from tqdm.notebook import tqdm, trange
from matplotlib.colors import Normalize
from scipy.stats import norm
from numba import njit, objmode
from mpl_toolkits import mplot3d
import random
import math
```

```
from numpy.linalg import norm
from matplotlib.offsetbox import AnchoredText
import matplotlib.patches as mpl_patches
from scipy import signal
import scipy.signal

obsdata = np.loadtxt("obsdata.txt", delimiter = ",") #mgal / cm/s^2 #doent work #w
mtrue  = np.array([780,930,1080,1200,1305,1410,1620,1695]).T
mtrueL = np.array([1,1,1,1,1,1,1,1])
mtrueU = np.array([1,1,1,1,1,1,1,1])*2500
mtrueU
NparamUsed = 4
mtrue  = mtrue[0:NparamUsed]
mtrueL = mtrueL[0:NparamUsed]
mtrueU = mtrueU[0:NparamUsed]
mtrueL[0:NparamUsed]
Nparam = 4
#make the function that calculates the misfit
def TestInvScat(model,model0,obsdata):

    Nparam = len(model)
    ref1 = np.zeros((10000))   #ref skal opdateret så hver iteration skubber disse
                               # så fordi vi får en ny raf hver gang, så vil wave

    for kk in range(0,3):
        n = np.array([1,2,3,4])
        ref1[model[kk]] += 0.02*(-1)**(n[kk]+1)



# BEGIN Initialize Basic Constants

    nsamp = 4096  # samples in seismogram

# Cosine-Gauss Wavelet with 3 major peaks

    N = 960 # 40;  # Number of samples of wavelet
    wav = np.zeros([N])
    T = 200 # 240; # Dominant period
```

```
    wav = np.cos((np.arange(-N/2+1, (N/2)+1))*(2*np.pi)/T) \
    *np.exp(-(1/2)*((np.arange(-N/2+1, (N/2)+1))**2)/(2*N))

    #calculate the waves
    ref = ref1[0:nsamp]
    #waves = np.convolve(ref,wav,'same')


    if np.logical_or((model == 1),(model == 2500)).any():
        waves = np.zeros(nsamp)
    else:
        npad = len(wav) - 1
        u_padded = np.pad(ref, (npad//2, npad - npad//2), mode='constant')
        waves = np.convolve(u_padded, wav, 'valid')




#find misfit

    stand = 0.10 # 0.02; # Small value -> target distribution closer to 0
    residual = np.zeros([nsamp,1])
    residual = waves - obsdata


    hx = (norm(residual[:],2)/stand)**2+(model-model0).T@(model-model0)/(150**2) #

    return hx
N = 120000
mod = np.zeros((Nparam,N)) # # Array with output model parameters #korrekt
mod[:,0] = mtrueL#mtrue # mtrueL #korrekt
mod[:,1] = mtrueU#mtrue # mtrueU #korrekt

tv = np.zeros([N]) #target value
tv[0] = TestInvScat(mtrueL,mtrue,obsdata)
tv[1] = TestInvScat(mtrueU,mtrue,obsdata)
sigma =40 #0.6 # 30 # 500 # 20 # 10 # 3 # 30 # 300
nacc = 2
rej=0
```

```
print("tv[0] {}".format(tv[0]))
print("tv[1] {}".format(tv[1]))
iterations = N
accepted_model = np.zeros(iterations)
log_L_mat = np.zeros((iterations))
for i in trange(2,iterations):
    prp = mod[:,i-1]
    prp = prp.astype(int)
    m_pert = -1
    u_z = np.random.uniform()


    rnd_pert = np.random.randint(1,5)

    #pertubation af model parameter 1
    if rnd_pert == 1:
        m_pert = mod[0,i-1]+(2. * u_z - 1.) * sigma #nu blivr den jo bare ved med
        if np.logical_or((m_pert > 1),(m_pert < 2500)).all():
            prp[0] = m_pert
        else:
            pass
    #pertubation af model parameter 2
    if rnd_pert == 2:
        m_pert = mod[1,i-1]+(2. * u_z - 1.) * sigma #
        if np.logical_or((m_pert > 1),(m_pert < 2500)).all():
            prp[1] = m_pert
        else:
            pass
    #pertubation af model parameter 3
    if rnd_pert == 3:
        m_pert = mod[2,i-1]+(2. * u_z - 1.) * sigma #
        if np.logical_or((m_pert > 1),(m_pert < 2500)).all():
            prp[2] = m_pert
        else:
            pass
    #pertubation af model parameter 4
    if rnd_pert == 4:
        m_pert = mod[3,i-1]+(2. * u_z - 1.) * sigma #
```

```
        if np.logical_or((m_pert > 1),(m_pert < 2500)).all():
            prp[3] = m_pert
        else:

    log_L_mat[i] = TestInvScat(prp,mtrue,obsdata)

    val =  TestInvScat(prp,mtrue,obsdata) #prp skal opdateres evt m_pert af alle 4
    p_acc = np.exp(-(1/2)*(val-tv[i-1]))

    u = random.random()#random.uniform(0,1)
    if u < p_acc:
        mod[:,i] = prp
        tv[i] = TestInvScat(prp,mtrue,obsdata)
        accepted_model[i]=1
    else:
        mod[:,i] = mod[:,i-1]
        tv[i] = tv[i-1]
    #print("mod[:,i] {}".format(mod[:,i]))
    print("prp {}".format(prp))

print("acceptance rate {}%".format(sum(accepted_model) /
                                   iterations * 100))
print("number of accepted models {}".format(sum(accepted_model)))
#hvad er mtrueU vs mt
mp_1 = mod[0,:][1000:-1]
mp_2 = mod[1,:][1000:-1]
mp_3 = mod[2,:][1000:-1]
mp_4 = mod[3,:][1000:-1]


f, ax = plt.subplots(1,1)
y, x, bars = plt.hist(mp_1,label="True = 780",bins=20, density=True, alpha=0.6,rwi
y1, x1, bars1 = plt.hist(mp_2,label="True = 930",bins=20, density=True, alpha=0.6,
y2, x2, bars2 =plt.hist(mp_3,label="True = 1080",bins=20, density=True, alpha=0.6,
y3, x3, bars3 =plt.hist(mp_4,label="True = 1200",bins=20, density=True, alpha=0.6,
plt.legend()
plt.title('Histogram of four sampled model parameters')
plt.xlabel('Model parameter value')
ymax = y.max()
```

```
idx = np.where(y == ymax)[0][0]
xval = x[idx]
xmin = np.round(x.max(),4)
# Annotate the highest value
plt.gca().text(xval, ymax, xval, ha='left', va='bottom',color='darkblue')
#
ymax = y1.max()
idx = np.where(y1 == ymax)[0][0]
xval = x1[idx]
xmin = np.round(x1.max(),4)
# Annotate the highest value
plt.gca().text(xval, ymax, xval, ha='left', va='bottom',color='orange')
#
ymax = y2.max()
idx = np.where(y2 == ymax)[0][0]
xval = x2[idx]
xmin = np.round(x2.max(),4)
# Annotate the highest value
plt.gca().text(xval, ymax, xval, ha='left', va='bottom',color='green')
#
ymax = y3.max()
idx = np.where(y3 == ymax)[0][0]
xval = x3[idx]
xmin = np.round(x3.max(),4)
# Annotate the highest value
plt.gca().text(xval, ymax, xval, ha='left', va='bottom',color='red')
mu1 = np.mean(mp_1)
mu2 = np.mean(mp_2)
mu3 = np.mean(mp_3)
mu4 = np.mean(mp_4)

box_style=dict(boxstyle='round', facecolor='wheat', alpha=0.1)
plt.figtext(1.12,0.7, "Sample results: $\mu_{1}$=%.2f," % (mu1), ha="center", va="
plt.figtext(1.12,0.6, "Sample results: $\mu_{2}$=%.2f," % (mu2), ha="center", va="
plt.figtext(1.13,0.5, "Sample results: $\mu_{3}$=%.2f," % (mu3), ha="center", va="
plt.figtext(1.13,0.4, "Sample results: $\mu_{4}$=%.2f," % (mu4), ha="center", va="


print("np.mean(mp_1) {}".format(np.mean(mp_1) ))
```

```
print("np.mean(mp_2)  {}".format(np.mean(mp_2) ))
print("np.mean(mp_3)  {}".format(np.mean(mp_3) ))
print("np.mean(mp_4)  {}".format(np.mean(mp_4) ))

plt.plot(log_L_mat[1:-1])
plt.xscale("log")
```

12.4 Informed proposal Monte Carlo Acoustic Wave

```
import numpy as np
import matplotlib.pyplot as plt
import numba as nb
from numba import jit
from tqdm.notebook import tqdm, trange
from matplotlib.colors import Normalize
from scipy.stats import norm
from numba import njit, objmode
from mpl_toolkits import mplot3d
import random
import math
from numpy.linalg import norm
from matplotlib.offsetbox import AnchoredText
import matplotlib.patches as mpl_patches
from scipy import signal
import scipy.signal

obsdata = np.loadtxt("obsdata.txt", delimiter = ",") #mgal / cm/s^2 #korrekt
refapprox = np.loadtxt("refapprox.txt", delimiter = ",")
plt.plot(refapprox)
plt.plot(obsdata)
mtrue  = np.array([780,930,1080,1200,1305,1410,1620,1695]).T #korrekt
mtrueL = np.array([1,1,1,1,1,1,1,1])
mtrueU = np.array([1,1,1,1,1,1,1,1])*2500
np.random.randint(1,5)
NparamUsed = 4
mtrue  = mtrue[0:NparamUsed]
mtrueL = mtrueL[0:NparamUsed]
mtrueU = mtrueU[0:NparamUsed]
Nparam = 4
```

```
#make the function that calculates the misfit
def TestInvScat(model,model0,obsdata):

    Nparam = len(model)
    ref1 = np.zeros((10000))   #ref skal opdateret så hver iteration skubber disse
                               # så fordi vi får en ny raf hver gang, så vil wave

    for kk in range(0,3):
        n = np.array([1,2,3,4])
        ref1[model[kk]] += 0.02*(-1)**(n[kk]+1)



# BEGIN Initialize Basic Constants

    nsamp = 4096  # samples in seismogram

# Cosine-Gauss Wavelet with 3 major peaks

    N = 960 # 40;  # Number of samples of wavelet
    wav = np.zeros([N])
    T = 200 # 240; # Dominant period
    wav = np.cos((np.arange(-N/2+1, (N/2)+1))*(2*np.pi)/T) \
    *np.exp(-(1/2)*((np.arange(-N/2+1, (N/2)+1))**2)/(2*N))

    #calculate the waves
    ref = ref1[0:nsamp]
    #waves = np.convolve(ref,wav,'same')


    if np.logical_or((model == 1),(model == 2500)).any():
        waves = np.zeros(nsamp)
    else:
        npad = len(wav) - 1
        u_padded = np.pad(ref, (npad//2, npad - npad//2), mode='constant')
        waves = np.convolve(u_padded, wav, 'valid')


    #convolution is the operation of taking two signals, inverting one of them,
```

```
    #and then shifting by multiplying with the value of the two signals and then s

#find misfit

    stand = 0.01#0.045 # 0.02; # Small value -> target distribution closer to 0
    residual = np.zeros([nsamp,1])
    residual = waves - obsdata


    hx = (norm(residual[:],2)/stand)**2+(model-model0).T@(model-model0)/(150**2) #

    return hx
N = 65000
mod = np.zeros((Nparam,N)) # # Array with output model parameters #korrekt
mod[:,0] = mtrueL#mtrue # mtrueL #korrekt
mod[:,1] = mtrueU#mtrue # mtrueU #korrekt

tv = np.zeros([N]) #target value
tv[0] = TestInvScat(mtrueL,mtrue,obsdata)
tv[1] = TestInvScat(mtrueU,mtrue,obsdata)
std_mp1 = np.std(np.array([777,772]))
std_mp2 = np.std(np.array([927,922]))
std_mp3 = np.std(np.array([1083,1088]))
std_mp4 = np.std(np.array([1197,1191]))
nacc = 2
rej=0
#vi skal bruge vores proposal q til vores p_acc
m0 = np.array([777, 927, 1083,1197])
Cm = np.eye(len(m0))*1*np.std(np.array([777, 772]))**2
Cm_inv = np.linalg.inv(Cm)

def log_q(mod,m0):
    dm = mod - m0
    return (-0.5 * (dm.T @ Cm_inv @ dm))

iterations = N
accepted_model = np.zeros(iterations)
log_L_mat = np.zeros((iterations))
for i in trange(2,iterations):
```

```
current_model = mod[:,i-1]
prp = mod[:,i-1]
prp = prp.astype(int)
m_pert_mp1 = -1
m_pert_mp2 = -1
m_pert_mp3 = -1
m_pert_mp4 = -1
if i == 2:
    log_L_mat[i] = TestInvScat(prp,mtrue,obsdata)
rnd_pert = np.random.randint(1,5)
if rnd_pert == 1:
    if np.logical_or((m_pert_mp1 > 1),(m_pert_mp1 < 2500)).all():
        u_m = np.random.uniform(-1,1)
        m_pert_mp1 = 777+u_m*12*std_mp1
        prp[0] = m_pert_mp1
    else:
        pass
if rnd_pert == 2:
    if np.logical_or((m_pert_mp2 > 1),(m_pert_mp2 < 2500)).all():
        u_m = np.random.uniform(-1,1)
        m_pert_mp2 = 927+u_m*12*std_mp2
        prp[1] = m_pert_mp2
    else:
        pass
if rnd_pert == 3:
    if np.logical_or((m_pert_mp3 > 1),(m_pert_mp3 < 2500)).all():
        u_m = np.random.uniform(-1,1)
        m_pert_mp3 = 1083+u_m*12*std_mp3
        prp[2] = m_pert_mp3
    else:
        pass
if rnd_pert == 4:
    if np.logical_or((m_pert_mp4 > 1),(m_pert_mp4 < 2500)).all():
        u_m = np.random.uniform(-1,1)
        m_pert_mp4 = 1197+u_m*12*std_mp4
        prp[3] = m_pert_mp4
    else:
        pass
if i > 2:
```

```
        log_L_mat[i] = TestInvScat(prp,mtrue,obsdata)
    log_q_curr = log_q(current_model,m0)
    log_q_pert = log_q(prp,m0)
    val =  TestInvScat(prp,mtrue,obsdata)

    p_acc = np.exp(-(1/2)*(val-tv[i-1]+log_q_curr-log_q_pert))

    u = random.uniform(0,1)#random.random()
    if u < p_acc:
        mod[:,i] = prp
        tv[i] = TestInvScat(prp,mtrue,obsdata)
        accepted_model[i]=1
    else:
        mod[:,i] = mod[:,i-1]
        tv[i] = tv[i-1]
    print("prp {}".format(prp))

print("acceptance rate {}%".format(sum(accepted_model) /
                                   iterations * 100))
print("number of accepted models {}".format(sum(accepted_model)))
#hvad er mtrueU vs mt
mp_1 = mod[0,:][1000:-1]
mp_2 = mod[1,:][1000:-1]
mp_3 = mod[2,:][1000:-1]
mp_4 = mod[3,:][1000:-1]

acc_rate = sum(accepted_model) / iterations * 100
f, ax = plt.subplots(1,1)
y, x, bars = plt.hist(mp_1,label="True = 780",bins=10, density=True, alpha=0.6,rwi
y1, x1, bars1 = plt.hist(mp_2,label="True = 930",bins=10, density=True, alpha=0.6,
y2, x2, bars2 =plt.hist(mp_3,label="True = 1080",bins=10, density=True, alpha=0.6,
y3, x3, bars3 =plt.hist(mp_4,label="True = 1200",bins=10, density=True, alpha=0.6,
plt.legend(loc='center left', bbox_to_anchor=(1, 0.8))
plt.title("4 Model Parameters: Acceptance rate = %.2f" % (acc_rate))
plt.ylabel("Sequence")
plt.xlabel('Model parameter value')
ymax = y.max()
idx = np.where(y == ymax)[0][0]
xval = x[idx]
```

```
xmin = np.round(x.max(),4)
# Annotate the highest value
plt.gca().text(xval, ymax, xval, ha='left', va='bottom',color='darkblue')
#
ymax = y1.max()
idx = np.where(y1 == ymax)[0][0]
xval = x1[idx]
xmin = np.round(x1.max(),4)
# Annotate the highest value
plt.gca().text(xval, ymax, xval, ha='left', va='bottom',color='orange')
#
ymax = y2.max()
idx = np.where(y2 == ymax)[0][0]
xval = x2[idx]
xmin = np.round(x2.max(),4)
# Annotate the highest value
plt.gca().text(xval, ymax, xval, ha='left', va='bottom',color='green')
#
ymax = y3.max()
idx = np.where(y3 == ymax)[0][0]
xval = x3[idx]
xmin = np.round(x3.max(),4)
# Annotate the highest value
plt.gca().text(xval, ymax, xval, ha='left', va='bottom',color='red')
mu1 = np.mean(mp_1)
mu2 = np.mean(mp_2)
mu3 = np.mean(mp_3)
mu4 = np.mean(mp_4)

box_style=dict(boxstyle='round', facecolor='wheat', alpha=0.1)
plt.figtext(0.92,0.6, "Sample results: $\mu_{1}$=%.2f," % (mu1), ha="center", va="
plt.figtext(0.92,0.5, "Sample results: $\mu_{2}$=%.2f," % (mu2), ha="center", va="
plt.figtext(0.93,0.4, "Sample results: $\mu_{3}$=%.2f," % (mu3), ha="center", va="
plt.figtext(0.93,0.3, "Sample results: $\mu_{4}$=%.2f," % (mu4), ha="center", va="
plt.tight_layout()


print("np.mean(mp_1) {}".format(np.mean(mp_1) ))
print("np.mean(mp_2)  {}".format(np.mean(mp_2) ))
```

```
print("np.mean(mp_3)  {}".format(np.mean(mp_3) ))
print("np.mean(mp_4)  {}".format(np.mean(mp_4) ))
#normal_mcmc = np.loadtxt("mcmcseis_burn_in.txt", delimiter = ",")
plt.plot(log_L_mat[1:-1],label="IPMC")
#plt.plot(normal_mcmc[1:-1],label="MCMC")
plt.legend()
plt.xscale("log")
plt.title("4 Model Parameters: Acceptance rate = %.2f" % (acc_rate))
plt.xlabel("Iterations")
plt.ylabel("Log(L(m))")
box_style=dict(boxstyle='round', facecolor='wheat', alpha=0.1)
CM = np.std(np.array([777, 772]))**2
stand = 0.045
apriori = 150
plt.figtext(0.8,0.6, "$C_{m}^{q}$=%.2f," % (CM), ha="center", va="center", fontsiz
plt.figtext(0.8,0.5, "$Noise$=%.2f," % (stand), ha="center", va="center", fontsize
plt.figtext(0.8,0.4, "$C_{m}^{prior}$=%.2f" % (apriori), ha="center", va="center",
plt.savefig('ipmc_burnin_4_mp.pdf')
```

12.5 Matlab De-convolution

```
function [refapprox] = decon(model,obsdata)
Nparam = length(model);

ref = zeros(10000,1);
for kk = 1:Nparam
    ref(model(kk)) = ref(model(kk)) + 0.02*(-1)^(kk+1); %gemmer reflektionskoeffic

end

%%% BEGIN Initialize Basic Constants %%%

nsamp = 4096;  % # samples in seismogram

% Cosine-Gauss Wavelet with 3 major peaks
N = 960; % 40;  % Number of samples of wavelet
wav = zeros(N,1);
```

```
T = 200; % 240; % Dominant period
wav = cos((-N/2+1:N/2)*(2*pi)/T).*exp(-(1/2)*((-N/2+1:N/2).^2)/(2*N));
%%% BEGIN Generate approximate model 'refapprox' from linear inversion %%%
% Compute simplified seismic data through convolution
a = zeros(1,nsamp);  % 'nsamp' is the length of the reflectivity
a(1:length(wav)/2+1) = wav(length(wav)/2:length(wav))'; % a is the "right half" of
b = a';
G = toeplitz(a,b);

ntimewin = 3900;
dobspad = zeros(nsamp,1);
dobspad(1:ntimewin) = obsdata(1:ntimewin);

eps2 = 0.5e-1;
refapprox = inv(G'*G + eps2*eye(nsamp,nsamp))*(G'*dobspad);

%%% END Generate approximate model 'refapprox' from linear inversion %%%



end
```