Monte Carlo methods for assessment of the probability distributions of sub sea resistivity models

by

Kristine Hermanrud

Thesis
for the degree of
Master of Science

(Master in Computational physics)

Faculty of Mathematics and Natural Sciences
University of Oslo

October 2009

Det matematisk- naturvitenskapelige fakultet
Universitetet i Oslo

Abstract

In this thesis we create a program which samples subsea resistivity models from the a posteriori probability distribution based on marine controlled source electromagnetic (CSEM) data.

In the first part we create a draft version of the code which assumes that any subsea resistivity model can be described by only three parameters. We use this code to identify the challenges we face when higher dimensional model spaces are allowed. The necessary improvements to the code is made and the final program developed.

The final program is then used to study the resolution power of marine CSEM data. The results confirm several resolution properties which have previously been identified using other approaches. In addition we find a probability measure describing the probability that certain resistivity properties are present in the model that produced a given dataset. Throughout these studies both synthetic cases and the Troll West Oil Province are under investigation.

Contents

Chapter 1

Introduction

The seismic-reflection method provides the highest resolution of subseafloor structure compared to other geophysical methods, and has therefore been the primary geophysical tool for hydrocarbon exploration. Seismic methods can help identify subsea properties such as pore fluid, lithology and porosity [1], but there is low confidence in the ability to determine whether a reservoir contains hydrocarbons or water [2]. A report by Thirud [3] states that as much as 90 % of all potential reservoirs are filled with saline water.

The marine controlled source electromagnetic (CSEM) method provides information about the subsea resistivity structure, a property separating water- from gas- and oil-filled reservoirs. Therefore, when the first field trials demonstrated the possibility of detecting hydrocarbon reservoirs prior to drilling [4] the marine CSEM method began to receive a significant amount of commercial interest [5].

In the marine CSEM method observations are made by collecting data from electromagnetic signals that have traveled through the seabed. The problem is then to find a hypothetical subsea resistivity structure that according to electromagnetic theory will provide data that fit the observations. To achieve this we need *Inversion theory* which describes how information about a parameterized system can be extracted from data observed from a measurement. The first formalizations of inversion theory dates back to 1760-1810. Two of the basic motivations was then to use astronomical data to infer the orbits of stellar objects and to use geodetic data to describe the shape of the Earth [6]. The solution to the inverse problem was then seen as the set of model parameters that best fit the observations.Since then advances have been made to the inverse theory, especially by geophysicists trying infer the interior of the earth.

Solving inverse problems is typically similar to the problem of finding a needle in a haystack. If the dimensionality of the problem is high then finding any model that fits the observations will require an extensive and time consuming search. Suppose that finding such a model is achievable,

then the difficulty of quantifying the non-uniqueness of the result arises:

How many models could describe the observations?

The main approach to reduce the number of solutions is to impose some *a priori* information on the models. Starting from some a priori state of information and taking the observations into account, we arrive at some *a posteriori* state of information.This a posteriori state of information can be described as a probability distribution over all possible subsea resistivity structures. This approach, which is applicable to any inverse problem, has previously been used for the purpose of inferring information about water-gas- and oil- saturation and porosity of the subsea floor, by using seismic amplitude versus angle (AVA) data and marine CSEM data jointly [7].

However, for the marine CSEM inversion problem where we infer information about the subsea resistivity, the a posteriori distribution of models is unexplored. Because marine CSEM inversion problems using 2.5D and 3D models are solved by gradient descent methods (Gauss-Newton and Quasi-Newton respectively), the model accepted as the solution is dependent on the start model and is likely to be a local optimum. Investigating the a posteriori probability distribution of 1D models should provide information about the non-uniqueness of the solutions and general resolution properties which are also relevant to the solutions to 2.5D and 3D inversion. The information we get from the a posteriori probability distribution can also be used to find a measure of the probability that the model that produced the observations contains specific resistivity values at depth.

The aim of this study is to create a program which samples subsea resistivity 1D models from the a posteriori probability distribution instead of finding one (possibly local) optimum, by using a Monte Carlo search consisting of a (pseudo-) random walk. This program will then be used to sample the a posteriori probability distributions using both synthetic and real marine CSEM data sets.

Chapter 2

Theory

In this chapter we introduce the basic theory applied in this thesis. First the principle of the marine CSEM method is explained. Then the focus is on inversion theory including probability theory and Monte Carlo methods. Finally the inversion theory and the marine CSEM method theory are merged to build the foundation this thesis is based on.

2.1 A short introduction to the marine CSEM method

In the marine CSEM method a mobile horizontal electromagnetic dipole (HED) is towed from a boat while emitting low frequency (<10 Hz) electromagnetic signals. These signals travel both through the seabed and the seawater. At the sea floor there is an array of electric field receivers. Electromagnetic energy constantly leaks from the seabed and is detected by the receivers [8].The electromagnetic signals detected by the receivers will be affected by the resistivity structure,and with the use of electromagnetic theory these signals are interpreted to infer subsea floor information. An illustration of a marine CSEM survey is found in figure 2.1.

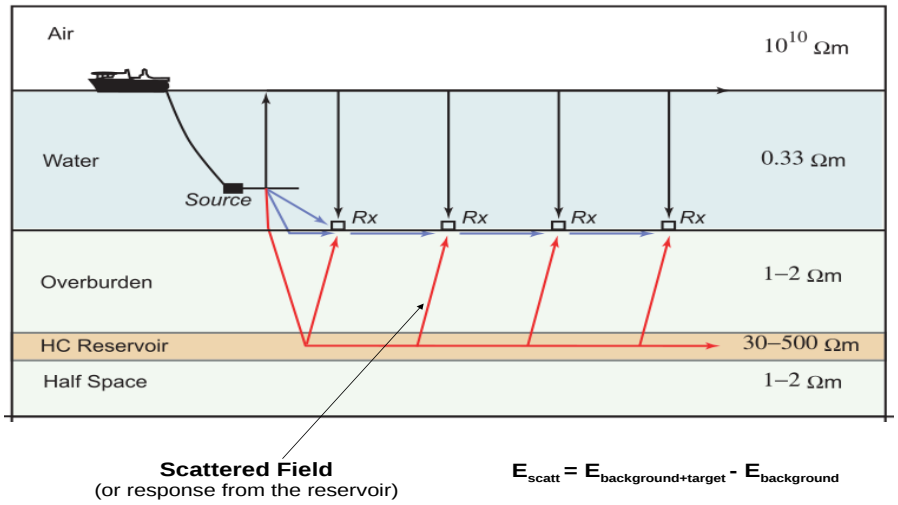
When an alternating electric current travels through a material, the amplitude will decrease with the propagation depth. The rate at which it decreases depends on the resistivity of the medium and the frequency of the wave. For a plane wave in a conductive medium the amplitude as a function of depth can be approximated by

$$E(d) \approx E_0(0) \cdot e^{-d\sqrt{\frac{\omega\mu\sigma}{2}}} = E_0(0) \cdot e^{-\frac{d}{\delta}} \tag{2.1}$$

where d is the distance, σ the conductivity, ω the angular frequency, μ the magnetic permeability and δ the *skin depth*.

The skin dept is also defined as *the distance a plane wave has propagated when the amplitude is reduced to* $\frac{1}{e}$ *of it's original value.* We see from equation 2.1 that high resistivity in a material and low frequency results in long skin

Figure 2.1 Typical marine CSEM inline survey



The electromagnetic fields detected by the receiver at the sea floor have followed different paths. The *direct signal* is the signal that has only propagated through the water and the *air wave* is the signal that has been guided through the air. The third possible path is through the seabed. The additional response due to a high resistive layer is the *scattered field*.

depth, and therefore little attenuation. Because the resistivity in water is in general lower than the resistivity in the seabed, the attenuation of the current is less in the seabed than in the overlying water column.Thus we expect that the signal measured by a receiver at a proper source-receiver separation primarily will consist of the components of the source fields which have followed a path through the seabed, and not of the fields that have traveled directly via the water. Knowing that hydrocarbon filled rock have high resistivity (in the range $10\Omega\text{m} \sim 200\Omega\text{m}$ whereas in water-filled rock and sand the resistivity is normally in the range $1 \sim 20\Omega m$), we expect that the amplitude of the source signal is less attenuated when propagating through hydrocarbons.

The orientation of the fields is also critical to the propagation behavior. If a conductive horizontal body is surrounded by less conductive matter, a current flow parallel to the boundary (i.e horizontal) will have continuous tangential electric fields across the surface, i.e,

$$E_1^T = E_2^T \tag{2.2}$$

where the subscripts denote the two media. This boundary condition will result in strong electric currents in the horizontal direction trough Ohm's law

$$J_i = \sigma_i \mathbf{E_i} \tag{2.3}$$

for $i = 1, 2$ where J_i is the current density, σ_i is the electrical conductivity in medium i. These strong electrical fields will induce magnetic fields through Amperes law:

$$\nabla \times \mathbf{H} = \mathbf{J}. \tag{2.4}$$

For a thin resistive layer however, induction alone will not be measurable [9] if the burial of the resistive layer below the measurement surface is much deeper than the thickness of the layer. Therefore thin, highly resistive targets are hard to detect using the magnetotelluric method [10].

To detect such thin, highly resistive anomalies in the seabed, vertical currents must be generated. If vertical currents or currents perpendicular to the boundary are generated, then the boundary condition becomes

$$J^N = J_1^N = J_2^N \tag{2.5}$$

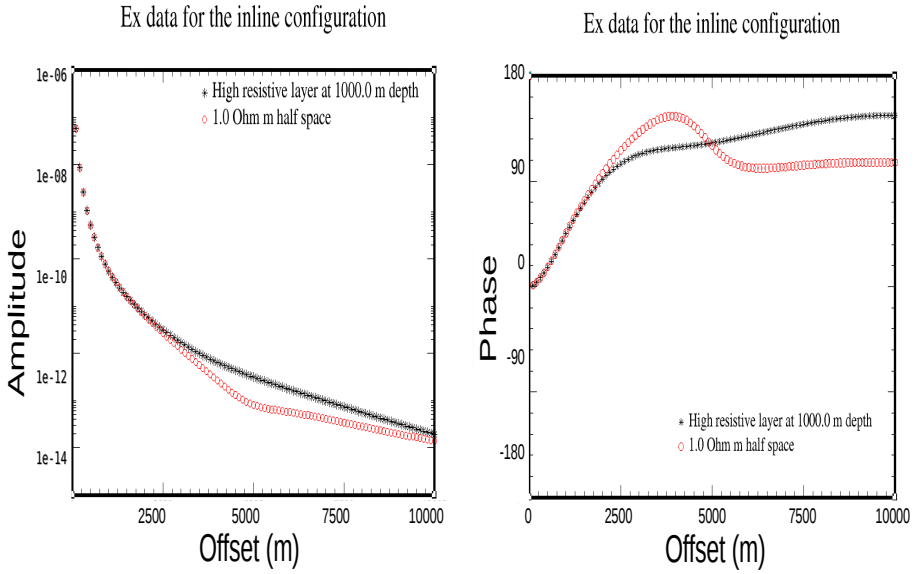
where the N superscript denotes that the direction of the currents are normal to the boundary.Together with Gauss' theorem this continuity condition results in a buildup of charge on the boundary [11]:

$$\rho_s = J_N \left(\frac{\epsilon_1}{\sigma_1} - \frac{\epsilon_2}{\sigma_2}\right) \tag{2.6}$$

where ρ_s is the surface charge, and ϵ is the dielectric permittivity. The charge buildup results in a galvanic effect that produces perturbations in the electric fields that are measurable at the sea floor.

The HED excites both vertical and horizontal components, which means that the response depends on the source-receiver geometry. The angle between the dipole axis and the line connecting the source and receiver is often referred to as the *source-receiver azimuth* and is used to describe the orientation of the fields. When the azimuth is 0^o, we have *in line geometry*. In this case, the components emitted by the source are mainly vertical and the response should be strongly affected by the presence of a hydrocarbon layer. If the azimuth on the other hand is 90^o we have *broadside geometry*, meaning the horizontal components will dominate.

Figure 2.2 Examples of amplitude and phase data



a) b)

For both plots a) and b) the red circles represents obtained data from a synthetic 1D $1.0\Omega m$ half space with 500 m water depth. The black starts represent the data when a 100.0 m thick highly resistive layer ($100.0\Omega m$) is introduced at 1000.0 m below the sea floor. The source frequency was 0.25 Hz. a) The amplitude vs. offset plots. b) Phase vs. offset plots.

The data, both amplitude and phase, at one receiver is highly dependent on the source-receiver separation also referred to as the *offset*. An example of the E_x data as a function of offset for an inline configuration is shown in figure 2.2. Here, the red circles represents the data obtained for a $1.0\Omega m$ half space under 500m water with resistivity $0.3125\Omega m$. The black stars represent the data when a 100m thick layer with resistivity $100\Omega m$ is

added at 1000.0 m below the sea floor. At 5000 m source-receiver separation, the amplitude plot shows a higher amplitude when the highly resistive layer is present. We also see that the phase is significantly affected by the highly resistive layer.

2.1.1 Electrical anisotropy

Electrical anisotropy occurs for thinly laminated sequences of sand and shale in sedimentary basins [12]. The effect of electrical anisotropy on marine CSEM data is seen in both phase and amplitude data. The degree of anisotropy can be characterized by the ratio

$$\lambda^2 = \frac{\rho_v}{\rho_h} \tag{2.7}$$

where ρ_v is the vertical resistivity and ρ_h is the horizontal resistivity. This complicates the interpretation of the inversion problem as the effect of electrical anisotropy can significantly influence the response measured at the sea floor. For instance, in reference [12] a highly resistive layer embedded in an anisotropic background gave a smaller *magnitude vs. offset* (MVO) response than if embedded in an electrical isotropic medium. The conclusion for the investigations in this paper was that a small reservoir was much more difficult to detect even for intermediate anisotropy ratios ($\lambda^2 = 2.0 \sim 3.0$).

2.2 Inverse problems

Feeding physical laws with parameter values and predicting the outcome is called *the forward problem*. The *inverse problem* uses the result of a measurement to infer the values of the parameters. The solution to the latter is typically non unique, meaning that more than one combination of parameters could have given the observed result, whereas the forward problem (in deterministic physics) has an unique solution.

2.2.1 Model- and data space

When a system has been parameterized, the *model space* $\mathcal{M}$ can be introduced, where each point $\mathbf{m} \in \mathcal{M}$, represent a conceivable model of the system. Each model consists of an ordered set of numerical values $\{m_1, ..., m_n\}$. In order to fully describe the system, the number of model parameters can either be finite or infinite.

When a parameterization has been chosen, each model, $\mathbf{m}$, can be represented by a particular set of values for the model parameters. The model parameters can either take on continuous or discrete values.

A basic definition used in inverse problems is the definition of probability distributions over the model space. The probability distribution is defined such that P($\mathcal{M}$) = 1 and P($\mathcal{A}$), the probability of $\mathbf{m} \in \mathcal{A}$, is a non-negative number when $\mathcal{A}$ is a subset of $\mathcal{M}$. This probability can be defined over any finite-dimensional model space.

The *data space* is the space of all conceivable instrumental responses. This data space can be denoted $\mathcal{D}$, and the result of one experiment represents a point **d** in data space [13]

2.2.2 Definition of probability

A process that selects points $\mathbf{m} \in \mathcal{M}$ is said to sample the *probability distribution P* if the probability that a chosen **m** is inside a region $\mathcal{A} \subseteq \mathcal{M}$ equals $P(\mathcal{A})$. Thus, if the process G samples P, we can investigate P by examining a set of points chosen by G, e.g. by drawing histograms representing the frequency with which G selects points inside suitable regions of $\mathcal{M}$.

Let $P(\cdot)$ be the probability distribution, then the Radon-Nikodym theorem states that there exists a function f(x) such that

$$P(\mathcal{A}) = \int_{\mathcal{A}} f(\mathbf{x}) d\mathbf{x} \tag{2.8}$$

where $\mathbf{x} = \mathrm{x}_1, \mathrm{x}_2, ..., \mathrm{x}_n$ and

$$\int_{\mathcal{A}} d\mathbf{x} = \underbrace{\int dx_1 \int dx_2 ..., dx_n}_{\text{over} \mathcal{A}} . \tag{2.9}$$

The function f(**x**) is the probability density [13].

2.2.3 A priori and a posteriori probability

The assumed probability distribution for all possible models is called the *a priori* probability distribution. When data from a measurement have been obtained, we can define the a posteriori probability distribution. This is a revised probability distribution that take into account the new information gained from the measurement [13, 14].

2.2.4 Joint, marginal and conditional probability

The probability of two or more events in conjunction is the *joint probability*. The *marginal probability* distribution of a subset of a collection of variables is determined from the joint probability density by integrating out the variables being discarded. The *conditional probability* is the probability of an event occurring given specific values for all the other variables [13].

2.2.5 Probabilistic Formulation of Inverse problems

In the following, let $\mathbf{d}_{cal} = d^1_{cal}, d^2_{cal}...$ be the calculated data for given values for the model parameters, or in other words, a solution to the forward problem for a given model. Furthermore, let $\mathbf{d}_{obs} = d^1_{obs}, d^2_{obs}...$ be the observed data from an actual measurement.Each model can be described with a set of model parameters as explained in section 2.2.1, $\mathbf{m} = \{m_1, m_2...\}$. Then, a general form of the forward problem can be written

$$\mathbf{d}_{cal} = g(\mathbf{m}) \tag{2.10}$$

where g(**m**) is the forward operator [14]. Because the inverse problem, (finding the model **m** having acquired a set of observed data values $\mathbf{d}_{obs}$), is usually both under determined and ill-conditioned, the question "What are the actual values of the model parameters ?" should be replaced by the question "What information can we infer on the actual values of the model parameters?". This leads to the Bayesian approach to inverse problems, which uses prior information and the information provided by a measurement to define the a posteriori probability.

The solution we end up with is of the form

$$\sigma(\mathbf{m}) = k\rho(\mathbf{m})L(\mathbf{m}) \tag{2.11}$$

[14] where $\sigma(\mathbf{m})$ is the a posteriori probability density, $\rho(\mathbf{m})$ is the a priori probability density and $L(\mathbf{m})$ is a likelihood function.

The discretized version reads

$$\sigma_i = \frac{\rho_i L_i}{\sum_j \rho_j L_j} \tag{2.12}$$

where $\rho_i = \rho(\mathbf{m}_i)\Delta m_1 \Delta m_2...$ is the equilibrium a priori distribution and $L_i = L_{\mathbf{m}_i}$

The term *likelihood* describes the hypothetical probability that an already occurred event would give a specific realization.

There are several ways of defining the likelihood function depending on the type of experimental uncertainties. If a vector of observed data, $\mathbf{d}_{obs}$,is described with Gaussian experimental uncertainties with a covariance matrix **C**

$$L(\mathbf{m}) = k \exp\Big(-\frac{1}{2}\Big[(g(\mathbf{m}) - d_{obs})^t \mathbf{C}^{-1}(g(\mathbf{m}) - d_{obs})\Big]\Big) \tag{2.13}$$

which for statistical independent experimental uncertainties degenerates into

$$L(\mathbf{m}) = exp\big(-S(\mathbf{m})\big) \tag{2.14}$$

where

$$S(\mathbf{m}) = \frac{1}{2}\sum_{i=1}^{N} \frac{(g^i(\mathbf{m}) - d^i_{obs})^2}{s_i^2} \tag{2.15}$$

is the misfit function and s_i^2 is the total noise variance.

The misfit function can be described in several ways, including as the sum of the absolute values of the misfit and as the sum of the squared absolute values of the misfit (as in equation 2.15).The fact that the misfit functions should depend on the experimental uncertainties became clear early in the history of inverse problems [6]. Inverse problems are often seen as an optimization problem (finding which model gives the lowest misfit) and the misfit function given by the squared absolute value of the misfit have been the most popular. The solution to this problem requires a least squares method which only involves the use of simple linear algebra.

2.2.6 Monte Carlo Methods

The term Monte Carlo method is often encountered in computational science, and can be described as a statistical simulation algorithm. Monte Carlo methods are widely applied in fields of natural science such as chemistry,physics, biology and medicine as well as in financial engineering [15] and econophysics [16].

Statistical simulation algorithms differ from numerical discretization methods in that instead of discretizing functions (typically partial differential equations) and then solving a set of algebraic equations, the solution is found by sampling from a probability distribution function (PDF) that describes the system. Monte Carlo methods are not only used for simulating stochastic processes, which can be described by PDFs, but also to solve problems with no apparent stochastic content. An example of this is the use of Monte Carlo methods for evaluating definite integrals. In this case the desired solution is imposed in the PDF and the simulation may be treated as a stochastic process. Multiple trials are carried out, and statistical properties such as averages and variances may be calculated. If the system can be described by PDFs, then the Monte Carlo simulation can proceed by randomly sampling from these PDFs. This requires a random number generator [17].

The sampling method that will be used in this thesis is the Metropolis algorithm.This is a Markov Chain Monte Carlo method where each step depends only on the previous step.The idea is to perform a random walk sampled from an initial probability distribution and then modifying this walk in such a way that the modified walk consists of samples from the target distribution. The Metropolis rule is the probabilistic rule that modifies the initial random walk. The classical Metropolis algorithm was introduced in reference [18] by Metropolis,and was used to sample the Gibbs-Boltzmann distribution describing the distribution of states in a system where

$$P_i = \frac{exp(-E_i/(k_B T))}{\sum_i exp(-E_i/(k_B T))} \tag{2.16}$$

where k_B is the Boltzmann constant, T is the temperature and E_i is the energy of the state i. The algorithm for sampling this distribution was as follows:

Starting at some initial state with energy E_{init} a perturbation in the system causing a change in energy ΔE is made. If this perturbation causes the energy of the system to become lower then this new state is accepted. Otherwise the move is allowed with the probability $exp(-\Delta E/(k_B T))$.

2.2.7 Sampling the a posteriori probability distribution using a Monte Carlo method

In section 2.2.5 an expression for the a posteriori distribution (see equation 2.11) was introduced. The question becomes:

How can we sample from this a posteriori distribution, $\sigma(\mathbf{m})$?

One way of achieving this is to sample from the a priori probability distribution and modify this distribution into the a posteriori distribution. To modify the a priori probability distribution, the likelihood function is used to decide if a model chosen according to the a priori probability distribution can also be accepted as a sample of the a posteriori probability distribution [14].

The recipe for sampling the a posteriori probability distribution is a follows: Suppose that L_j is the value of the likelihood function for a particular model $\mathbf{m_j}$. A perturbation on this model is chosen from the a priori probability distribution giving a model $\mathbf{m}_i$ which gives a value for the likelihood function L_i. Then the rule for accepting or rejecting a model is as follows.

1.If $L_i \geq L_j$ (i.e the new model has higher likelihood than the old model) then accept the move

2. $L_i < L_j$ (i.e., the new model gives a lower value of the likelihood function than the old point) then accept the move with probability L_i/L_j

It is shown in reference [14] that this sampling algorithm produces samples from the a posterior distribution. To see the connection between this sampling method and the metropolis algorithm, let us insert the expression for the likelihood function given in equation 2.14 in the sampling method described above. The acceptance rule then becomes

$$P_{accept} = \begin{cases} 1 & \text{if} S(\mathbf{m}_{new}) \leq S(\mathbf{m}_{old}) \\ exp(\frac{-\Delta S}{s^2}) & \text{if} S(\mathbf{m}_{new}) > S(\mathbf{m}_{old}) \end{cases} \tag{2.17}$$

where $\Delta S = S(\mathbf{m}_{new}) - S(\mathbf{m}_{old})$. With a uniform a priori distribution this becomes identical with the metropolis algorithm in section 2.2.6.

An important feature of this method is that it is independent of the way the probabilities have been normalized, meaning that the relative probabilities of the models can be inferred from the random walk even before the

walk has been extensive enough for the denominator in equation 2.12 to be calculated with good precision.

2.3 Theory applied to the marine CSEM method

The model space in marine CSEM inversion may consist of model parameters representing the resistivity as a function of depth below the sea floor, $\rho(z); z \in (0, a)$ where $m(z) = \rho(z)$ and a is the maximum depth of investigation. This gives an infinite number of parameters for each model. However, the space can be discretized, $\rho_\alpha = \rho(z_\alpha)$ where $\alpha = 1, ..., n$, and a finite number of parameters can then be used in the inversion [13]. The allowed values for the model parameters,i.e the resistivity, are continuous because the resistivity in the sea bed at a given depth can be any real number (within reasonable limits).

The data space in the marine CSEM method consists of all conceivable electromagnetic responses detected by the receivers at the sea floor.

The joint probability will in this case be the probability that a specific model (with given values for all resistivity parameters) is equal to the model that produced the set of obtained data. Because our model space is multidimensional, visualizing the joint probability distribution is highly impractical. Instead, we can plot the marginal probability distributions of models. This way the number of dimensions to represent is reduced. However, information included in the joint probability distribution will be lost by marginalizing the joint probability distribution.

The inversion problem in the marine CSEM method is finding a resistivity (or conductivity) profile of the subsurface that gives a good data fit with the actual measured data. Several methods for finding a misfit minimum like Newtons method, Gradient descent and Simulated annealing, are commonly used for inverting CSEM surveys. The solution obtained by inversion is, as mentioned in section 2.2 non-unique, which means there may be several equally good minima. In the general case there is also a multitude of sub-optimal minima. The methods mentioned above all run the risk of ending up in a local minimum instead of a global minimum. This is especially the case for Newtons method and gradient methods. The simulated annealing scheme will, on the other hand, find the global misfit minimum if the cooling of the temperature is infinitely slow. For a high dimensional inverse problem many iterations are required to find the global minimum.

To exclude sub-optimal solutions it is common to constrain the models with some a priori information. This information could be in the form of hard or soft constraints. A hard constraint is a direct constraint on the parameters. An example could be that the resistivity in a particular layer never exceeds a given value. A soft constraint is imposed as a penalty in the

misfit calculation. This type of constraint ensures that models that are seen as less physical according to our a priori information, give larger misfit.

Assuming independent Gaussian distributed uncertainties, the misfit function in equation 2.14 can be written

$$S(\mathbf{m}_i) = \sum_x \frac{|E(x)^{\text{obs}} - E(x)^{\text{synt}}|^2}{\alpha^2 |E(x)^{obs}|^2 + \eta(x)^2} \tag{2.18}$$

where the superscripts obs and synt referrers to the observed data from the *reference model* and data from the synthetic sample models respectively. The reference model is the actual model from which the obtained data were acquired. Here, α is the multiplicative uncertainty in the acquired data and η represents the additive background noise. The multiplicative data uncertainty is typically 1-5% of the amplitude. The cause of this data uncertainty can be bathymetry, slight rotations of the source etc. Then the measurement uncertainty is proportional to the measured value. In contrast, the additive data uncertainty does not depend on the measured signal. However, this uncertainty may depend on acquisition parameters such as the source frequency and position. The background noise is due to electrical signals that do not contain information about the seabed, and is either caused by the instrumentation or any other external uncontrolled source. An example of background noise encountered in the marine CSEM method is the natural field emissions that arise from the interaction of the solar wind with the Earth's magnetosphere.

When the background noise is high compared with to the data it may be better to discard the measurement. This can be done by removing the denominator in equation 2.18 and instead multiplying the equation with weights w_x given by

$$w(x) = \begin{cases} 0 & \text{if} \frac{|E(x)^{obs}|^2}{|\eta(x)|^2} < 10 \\ \frac{1.0}{\alpha^2 |E(x)^{obs}|^2 + \eta(x)^2} & \text{otherwise} \end{cases} \tag{2.19}$$

where η is the noise level.

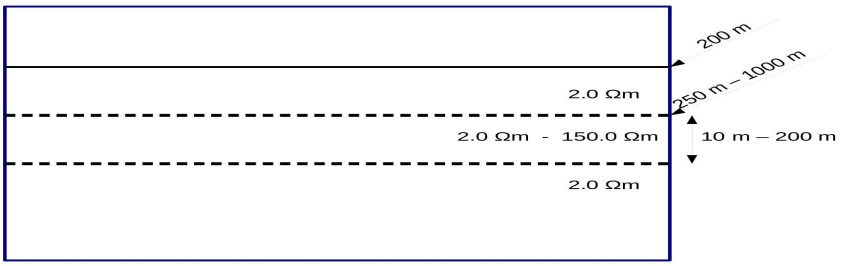
Chapter 3

Method development

In this chapter we focus on the development of the source codes used to produce the results in chapters 4 and 5.First we conduct a simple synthetic study where each subsea resistivity model is fully described by three parameters. The results from this study are then used to develop a code that handles higher dimensional model spaces. Finally, we describe the resulting algorithm used to produce the results in chapters 4 and 5, and highlight some questions that can be answered by applying this method.

3.1 The first tests

Figure 3.1 Illustration of all possible models $\mathbf{m} \in \mathcal{M}$



All models $\mathbf{m} \in \mathcal{M}$ consist of three horizontal, plane layers beneath the sea floor. The water resistivity is fixed at $0.3125\Omega m$ and the resistivity in the top and bottom layers are fixed at $2\Omega m$. The water depth is 200m. The parameters that are varied in this Monte Carlo simulation are the resistivity, thickness and depth of the middle layer.

A python script was made for the first numerical experiments using Monte Carlo simulations to investigate the a posteriori probability distributions of sub sea models. In addition to assuming only one high resis-

tive horizontal layer the code also assumes known water depth and uniform background resistivity. There are only three parameters to vary in this Monte Carlo simulation: resistivity, depth and thickness of the high resistive layer. An illustration of all possible models is shown in figure 3.1.

Prior to starting the Monte Carlo simulation, the script takes as input values limits for the model parameters (resistivity, depth and thickness of the high resistive layer), water depth, the file containing data from the reference model etc. The input parameters that are not specified by the user is set to default values.

When all initializations are made, a start model is created by randomly choosing values for the model parameters within the user specified limits. Then, data from this start model is calculated by solving the forward problem [1]. This data is then used to calculate the misfit,$S(\mathbf{m}_{start})$, with the data obtained for the reference model and the value for the likelihood function $\exp(-S(\mathbf{m}_{start}))$, where $S(\mathbf{m}_{start})$ is given by equation 2.18.

At this stage,the current model is set to equal the start model and the Monte Carlo loop begins. The current model is perturbed by varying one of the model parameters, and the likelihood value of the new model is calculated. This likelihood value is then compared with the likelihood value obtained for the current model. The perturbed model is accepted or rejected as a sample of the a posteriori probability distribution according to the algorithm described in section 2.2.7. If the sample is rejected, nothing happens and the loop continues by perturbing the current model again and testing the new model for acceptance. Otherwise, if the sample is accepted, the current model is set equal to the accepted model, and the operations described above are repeated. The loop continues until some stopping criteria are met.

The perturbations in the Monte Carlo loop are done by varying each model parameter in turn. The first variation is on the resistivity of the high resistive layer. Regardless of whether the model is accepted or rejected, the next perturbation is made by a variation on the depth of the high resistive layer. Then, the variation is on the thickness, followed by a variation in resistivity, and so on.

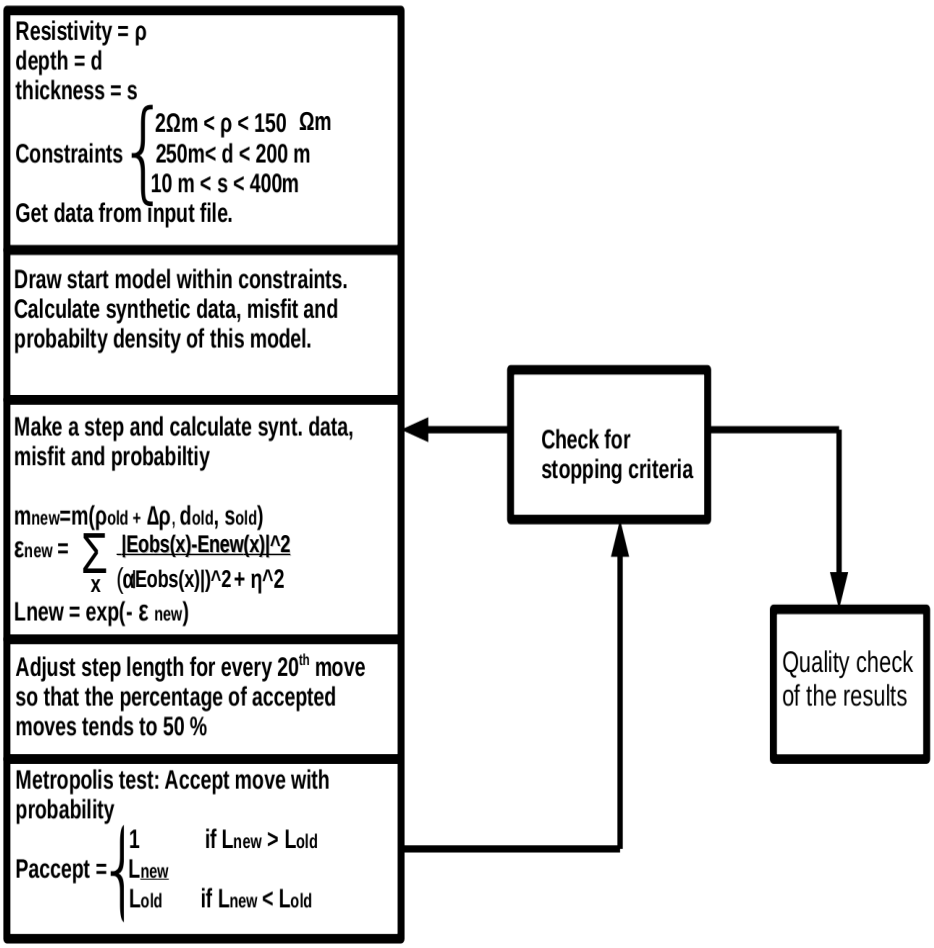
During these first tests a uniform a priori probability distribution was used. This means that the only a priori information assumed about the sub sea resistivity structure is that all resistivity values are equally probable. In this case the a posteriori probability distribution simply becomes

$$\sigma(\mathbf{m}) = kL(\mathbf{m}). \tag{3.1}$$

The flow chart for the the script described here is found in figure 3.2 and the script can be found in appendix B.1

[1] This is done by a using an existing modeling code for plane 1D earth models called *elcardinal* developed at EMGS

Figure 3.2 The flow chart for script used in the first numerical tests



First a start model within the constraints given by the user is constructed. Then, one of the model parameters (resistivity, depth or thickness) are varied. Then, the value of the likelihood function is calculated for the new model and accepted or rejected by the metropolis rule. These actions are then repeated until the stopping criteria are met. The step lengths are updated as the simulation proceeds such that the number of accepted moves tends to 50 %

3.1.1 Description of the reference model and data acquisition

The reference model used here consists of a single high resistive layer of $80\Omega m$ in a uniform background with resistivity $2.0\Omega m$. The highly resistive target is placed at 500m - 600m below the sea floor (which is shallow), and the water depth is 200.0 m (also shallow). The resistivity of the water is $0.3125\ \Omega m$. Inline geometry is assumed for the acquisition, so that galvanic effects dominate (see section 2.1). The receiver spacing is set to be 100 m. We invert for one receiver where the first source-receiver offset is at 0.0 m and the last offset is at 10 km.

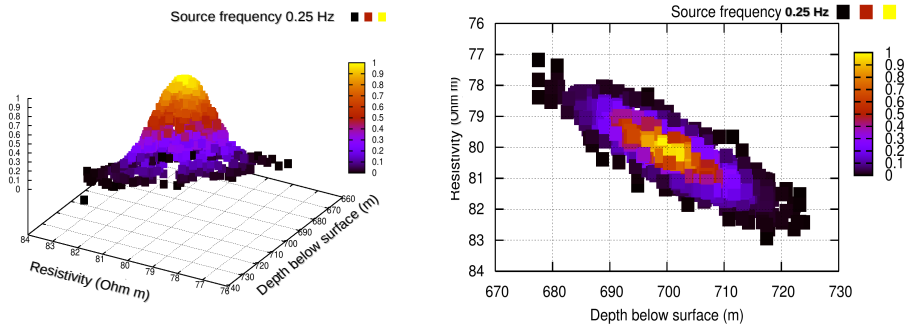
3.1.2 Some preliminary results

The script was run three times using three different source frequencies for the synthetic data acquisition. The non-normalized a posteriori probability distributions (which are here equal to $L(\mathbf{m})$) for the Monte Carlo simulation are shown in figures 3.3 and 3.4. In both figures, one thousand samples of the a posteriori probability distribution were discarded in the beginning of the Monte Carlo simulation to ensure a collection of samples from the area of interest.

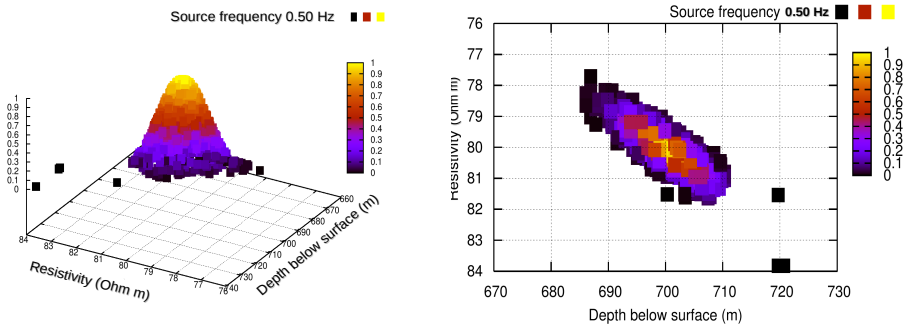
In figure 3.3 the thickness of the reservoir was held fixed at 100 m while the resistivity and depth of the target layer could vary. The likelihood values are represented by the color scale. We see that for all three source frequencies, 0.25 Hz, 0.50 Hz and 0.75 Hz, the resistivity and depth values representing samples of the a posteriori probability distribution are concentrated around the point $80\Omega m$ and $700.0m$ below the surface. At these values, which are the same values as for the reference model, we also have the maximum likelihood values.

The plots in figure 3.4 show non-normalized a posteriori distributions when keeping the depth of the reservoir fixed at 700 m below the surface varying only the thickness and resistivity of the target layer. In figure 3.4a a source frequency of 0.25 Hz was used and in figure 3.4b a source frequency of 0.75 Hz was used. Here too, the likelihood values are represented by the color scale, and the resistivity and thickness are displayed on the x- and y-axis respectively. In this case we see that the a posteriori distribution of models looks very different. The most characteristic feature in these two figures is the strong correlation between the thickness and the resistivity. In the box $74\Omega m < \rho < 84\Omega m$ and $94m <$ thickness $< 110m$ the figures 3.4a and b are almost identical. Very few samples were sampled outside of this box. The shape of the a posteriori distributions resembles the shape of the function $f(x) \propto \frac{1}{x}$. Also, we see that the point with resistivity of $80\Omega m$ and thickness of 100 m lies on this curve.

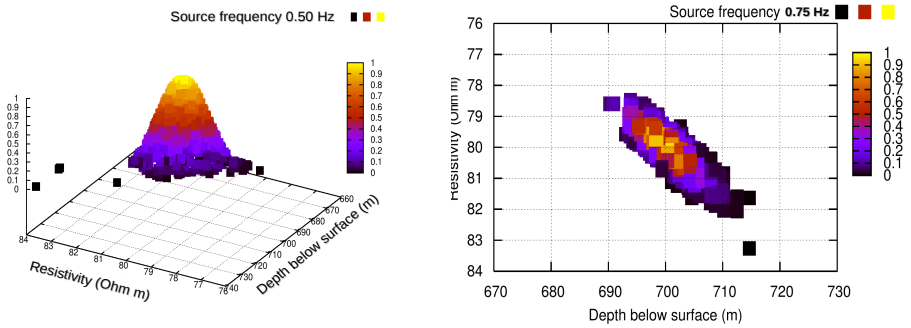
Figure 3.3 Samples of the a posteriori probability distribution



a) Frequency: 0.25 Hz



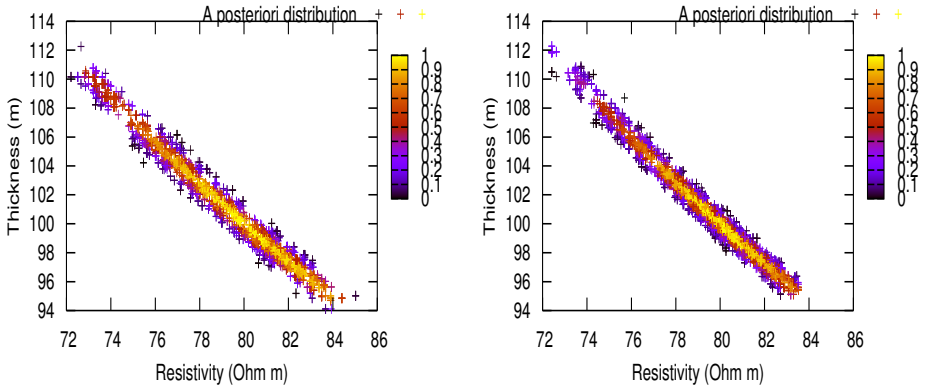
b) Frequency: 0.50 Hz



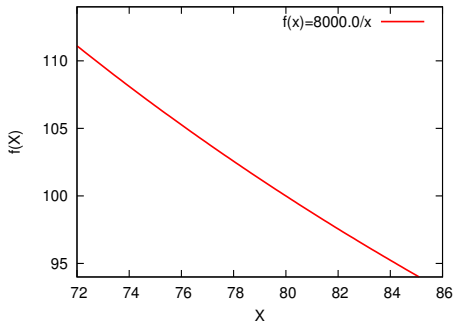
c) Frequency: 0.75 Hz

The value of the likelihood function for each model is represented by the colors in these plots. Warmer color means higher likelihood. The data uncertainty in these plots were assumed to be 5 % and the background noise was assumed $\eta = 10^{-15} V/Am^2$. Whatever the frequency used in the synthetic acquisition, the a posteriori distribution of models show a Gaussian like function. The maximum likelihood is found at the values defining the reference models. All simulations started from the same start model.

Figure 3.4 Plots of resistivity vs. thickness



a) Frequency = 0.25 Hz b) Frequency = 0.75 Hz



c) $f(x) = \frac{8000.0}{x}$

a) and b) show the a posteriori probabilities when the depth is held fixed at 700.0 m below the surface. The plot a) was the result when using a 0.25 Hz source frequency whereas b) was produced using a 0.75 Hz source frequency. In b) the area that the random walker has sampled is slightly smaller than in a). However the two plots are almost identical,and the function $f(x) = \frac{8000.0}{x}$ in c) fit both the a posteriori distributions in a) and b). Thus, there is a strong correlation between the thickness and the resistivity. This correlation does not seem to be strongly frequency dependent

3.1.3 Discussion and conclusions for the first tests

The main question that presents itself in the results from section 3.1.2 is

> Why do we have the strong correlation between thickness and resistivity?

The correlation between the thickness and resistivity of the target indicates that the total resistivity of the models is an important quantity. If the layer is thinner than the reference value, then the resistivity in the layer is higher and vice versa.The function $f(x) = \frac{8000.0}{x}$ fit both the curves in figure 3.4a and b. The function $f(x) = \frac{8000.0}{x}$ is displayed in figure 3.4 c for comparison. Thus, the "total" resistivity $\int_z \rho(z)dz$ is very well resolved here. For further investigations we should therefore save the integral value $\int_z \rho(z)dz$ for each sample of the a posteriori distribution and not only the resistivity values.

We also notice that the higher the source frequency, the narrower the a posteriori distribution of models in figure 3.3. A narrow a posteriori distribution represents well resolved values. In addition we see that the resolution improvement due to using higher source frequency is larger when using 0.50 Hz instead of 0.25 Hz than it is when using 0.75 Hz instead of 0.50 Hz. However, the shape of the a posteriori distribution in figure 3.3 b and c differ significantly. When using a source frequency of 0.50Hz, the distribution has a tail of models with low likelihood values towards shallow depths and low resistivities. For the 0.75 Hz case, the a posteriori distribution has a tail representing models with low likelihood values towards deep depths and high resistivities. We conclude that for further tests we should pay extra attention to the effect of using different frequencies when acquiring data, and try to understand the physics explaining the results.

3.2 Preparing the multilayer Monte Carlo code

When producing the figures in section 3.1.2 we calculated data and the data misfit for around nine thousand models. The computation time was around twenty minutes for each result. When moving to a higher dimensional problem (multilayer models), the size of the model space increases dramatically. Suppose our discretization of a model consists of n layers, then the dimensionality of the model space is proportional to n. To sample the a posteriori probability distribution over a large model space will require a significantly increased number of model samples. This could create problems for further investigations.

To prepare a code that will work for a higher dimensional model space some modifications of the present script should be made. We need to identify the problems that may occur when moving to a higher dimensional

model space before we can suggest improvements. First we want to answer the questions

1) Which parts of the code are the most time consuming

and

2) How much time should we expect to need for higher dimensional problems?

The part solving the forward problem (i.e. running elcardinal) is by far the most time consuming part. Running elcardinal for three layered models two hundred times took 23.24 seconds, while running the whole script including allocations calculations, reading and writing to files for two hundred models took 25.67 seconds.

However, when moving to higher dimensional model spaces, the time spent on one forward calculation also increases dramatically. In the present script only three resistivity values at different depth intervals are given as input to elcardinal. We hope to have a code that can handle at least fifty resistivity values at different depth intervals describing the subseafloor. Two hundred forward computations using thirty layers, where each layer had one resistivity value, took 95.15 seconds. This is more than four times the time it took for the forward computations using only three resistivity values.

Suppose we want to use the same number of *whole iterations* as for the previous tests where a whole iteration means that all parameters have varied once. Then we would need 10 times the number of forward computations compared to when varying only three parameters. Also, each forward calculation takes four times as much execution time. Instead of using 20 minutes we would now need 13 hours.

Taking into consideration that 3000 whole iterations will not be nearly enough to sufficiently sample the a posteriori probability distribution and that thirty parameters is a low number of parameters, we understand that some improvements has to be done. Reducing the number of forward calculations would improve the computational time significantly.

3.2.1 Speeding up the code

One way of reducing the number of forward computations is to calculate the Frechet derivatives of the E-field with respect to the conductivity and use these derivatives to estimate the data in a new point $\sigma + \Delta\sigma$. By doing this we avoid calling elcardinal every time we wish to calculate the value of the misfit function. We estimate the data in the new point in model space,$E_x(\text{offset}, \sigma + \Delta\sigma)$, by

$$E_x(\text{offset}, \sigma + \Delta\sigma) \approx E_x(\text{offset}, \sigma) + \frac{\delta E_x(\text{offset}, \text{œ})}{\delta\sigma}\Big|_d \Delta\sigma \qquad (3.2)$$

where d is the depth and $\frac{\delta E_x(\text{offset},\sigma)}{\delta\sigma}$ is the Frechet derivative. By Taylor expansion the error due to this approximation is $\propto \frac{\delta^2 E_x}{\delta\sigma^2}\Big|_d \Delta^2\sigma$ which is small for small $\Delta\sigma$.

The error in the misfit calculation due to estimation of data using the Frechet derivatives will depend on the step length and the depth of where the conductivity perturbation takes place. In figure 3.5 the step length- and depth- dependence of the relative error in the misfit calculation is displayed. A half space model consisting of 500 m water and 2.0 Ωm resistivity in the seabed was used as reference model. The model that was perturbed was a half space model of 500 m water depth and a resistivity of 1.0Ωm in the seabed. In each layer (100 m thick), the resistivity was perturbed by a fixed step length. The relative error,

$$Err = \frac{S_F(\mathbf{m}) - S_E(\mathbf{m})}{S_E(\mathbf{m})} \tag{3.3}$$

was calculated for all the perturbations. Here $S_F(\mathbf{m})$ and $S_E(\mathbf{m})$ denotes the misfit in equation 2.18 when the Frechet derivatives and elcardinal respectively was used to calculate the response from model $\mathbf{m}$. Then the step length increased, and the conductivity values in all layers were perturbed with this new step length.

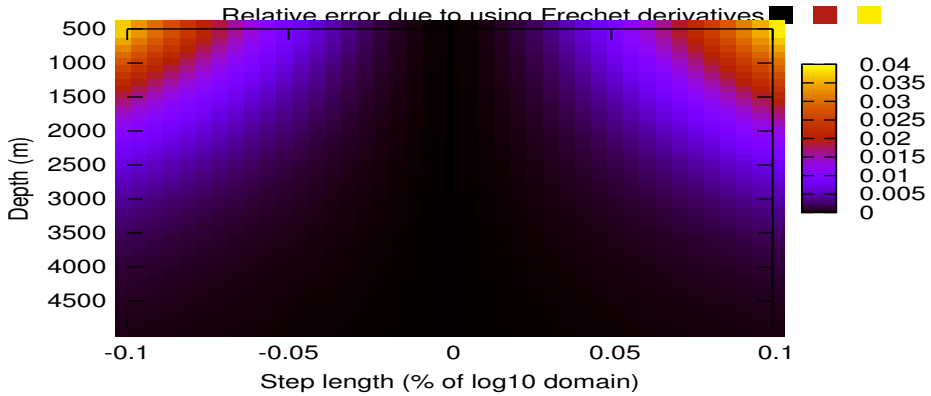
We see that small perturbations in deep layers result in low errors, while large perturbations in shallow layers give high errors. The error will also depend on $\frac{\delta^2 E(\sigma)}{\delta\sigma^2}|_d$ which in turn will depend on the model under consideration. Thus, it is difficult to find concrete rules, such as limiting the step length at each depth interval, which ensures a small error in the misfit calculations.

Hopefully, updating the step lengths in such a way that the acceptance rate in each layer tends to 50 % will indirectly ensure small misfit errors: The sensitivity to a change in resistivity in shallow layers is presumably in general higher than in deep layers. Thus, the step length should automatically end up being smaller in the top layers. However, to make sure that we do not accept models with high misfit error, we can check the error before saving a model as a sample of the a posteriori distribution. We can then exclude the samples with high error.

Another way of speeding up the code significantly is to use parallel computing. This means that we execute a number (f.ex 50) of jobs at the same time (in parallel), and collect the results when all jobs have finished. Roughly, if we would like to sample 100 000 models using 50 processes, the execution time will be the same as if 2000 models were sampled using only one process. [2] When parallel computing, there are a few things to consider.

[2]Tho finish the whole experiment will take a bit more time than the estimate in the text because collecting the results from the processes takes some additional time.

Figure 3.5 Relative misfit error



The reference model was a half space model with background resistivity of $2.0\Omega m$. The test model used to calculate the relative error is a half space of $10.0\Omega m$, and perturbations on this test model were made to estimate the data using the Frechet derivatives. This data was used to calculate the relative error in the misfit, $Err = \frac{\text{Misfit}_F - \text{Misfit}_E}{\text{Misfit}_E}$. Here, Misfit_E is the data misfit with the reference model calculated when elcardinal was used to estimate the data and Misfit $_F$ is the data misfit when the Frechet derivatives were used to calculate the new data.

The random number generator needs an integer starting value (seed) to generate a sequence. If the seed is not specified, then Python automatically sets the seed based on the system time, which will give a different sequence at different times. However, when submitting jobs simultaneously, the seed might be identical for all the jobs and the generated numbers will be the same. [19] This will result in the exact same samples of the a posteriori probability distribution for every job, and the whole point of parallelizing the code vanishes. It is therefore important to specify different seeds for each job.

All of the jobs start from the same start model. This start model may or may not be a model with good misfit. If the start model is far from a misfit minimum, then it will take many iterations before the random walker starts to sample around the parts in model space with relatively high likelihood. It becomes even more important to skip a sufficient number of the first iterations when all jobs start from the same start model. Otherwise, the start model and the first samples of models close to the start model will give a significant but misleading contribution to the a posteriori probability distribution. The question becomes

How many iterations should we skip?

One way of determining this is to calculate the average value for the resistivity in each layer and wait with saving the models until the average has stabilized for every layer.

3.2.2 Using a Simulated Annealing algorithm to find a start model

The problem with a large model space, is the large number of models far from the reference model. This means that the chance of starting at a position in model space with a high likelihood value gets very small. Using a Markov chain Monte Carlo method as a sampling method requires that the starting position is close to the area of interest. Otherwise, the random walker can get stuck in a place far from the true model, and we would never know of the area of much better data misfit. A solution to this is to first try to find a misfit minimum, and then use this as a starting position for the Monte Carlo sampling.

The method *Simulated Annealing* is a good method to finding a start position. The method was invented in the 1980's [20] and has become a tool in geophysical problems and has successfully been applied to marine CSEM studies [21]. Simulated annealing is a numerical method used to solve

$$\min\{f(x)|x \in D\}$$

where D is the domain. The idea is to start at some position $x = x_0$ and take a (small) random step $x_1 = x_0 + \Delta x$. If $f(x_1) < f(x_0)$ then the move

is accepted (x is set equal to x_1). Otherwise, the move is accepted with the probability $e^{-\Delta x/T}$ where T is some constant referred to as the temperature. Then the procedure is repeated by taking another random step from the new x. The loop continues for a given number of iterations before T is lowered, meaning that the probability of accepting a "bad" move decreases. The loop then continues using the new, lower temperature. The difference between this algorithm and the Monte Carlo sampling with the Metropolis test lies in the lowering of the temperature which ensures that the random walker will stagnate in a (hopefully global) minimum.

3.2.3 Sampling

Because both vertical and horizontal components are emitted by the HED the data are sensitive to both highly conductive and resistive layers (see section 2.1). If the step lengths taken in the Monte Carlo simulation are uniform with respect to resistivity then the step lengths with respect to conductivity are highly non uniform. By introducing the parameters $\rho' = log10(\rho)$ and $\sigma' = log10(\sigma)$ then

$$\sigma' = log10(\sigma = \frac{1}{\rho}) = -log10(\rho) = -\rho' \tag{3.4}$$

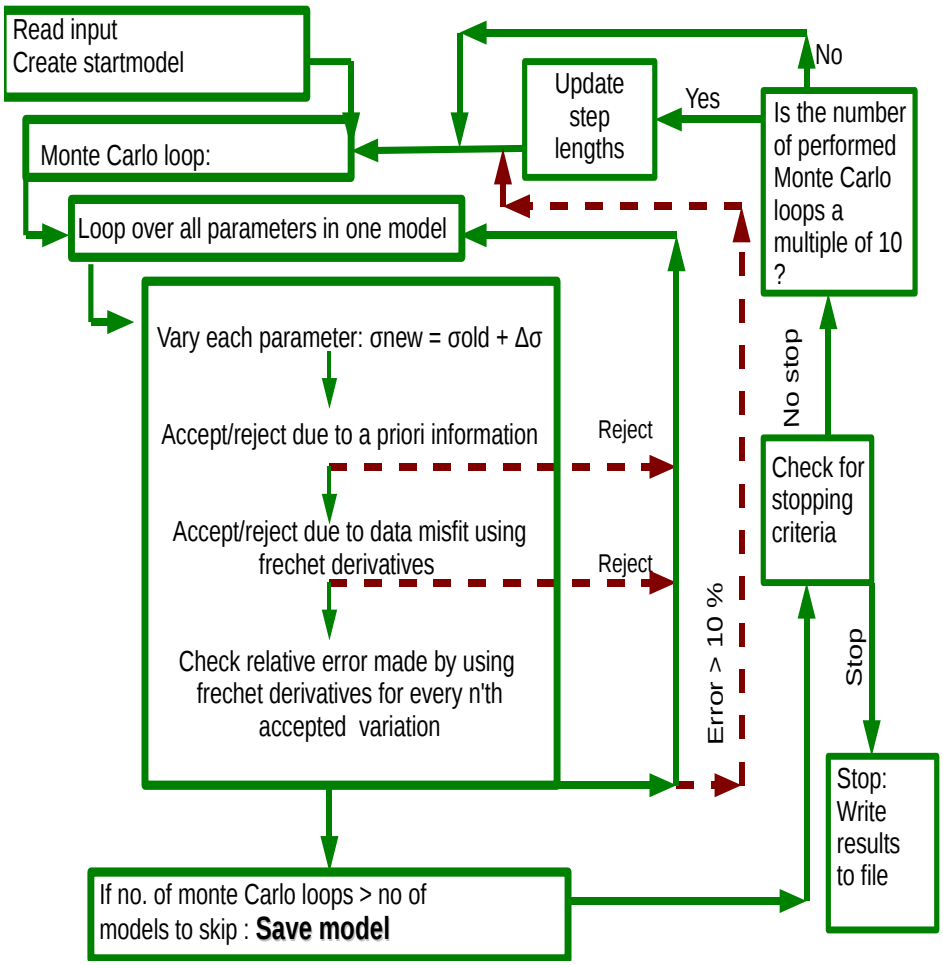
and a uniform sampling in the log10(ρ) scale will give equally large steps with respect to both ρ' and σ'.

3.3 Description of the source code

The idea behind the multilayer Monte Carlo code is simple; keeping the layers fixed in depth and thickness, the only attribute that varies is the resistivity.

In figure 3.6 the flow chart for the multilayer Monte Carlo code is given. First all the input parameters such as maximum value for the conductivity in each layer, the input files (with the observed data and noise), number of layers etc. are initialized. Then, the weights in equation 2.19 are calculated and a start model is created. The start model can be explicitly chosen by giving the file containing the start model as input. If the start model is not given by an input file, a half space model,i.e. a model with uniform resistivity in the seabed,is created. After the initializations, the Monte Carlo loop begins by first perturbing the resistivity value in the shallowest seabed layer. Then the metropolis test based on the specified a priori information alone checks if this perturbation should be accepted. If the move is rejected, the loop continues by varying the second shallowest layer. Or if the move is accepted as a sample of the a priori distribution, a new acceptance test based on the data misfit is performed. If the move is accepted (passes both

Figure 3.6 Flow chart for the multilayered Monte Carlo script



the acceptance tests), then this new model with the perturbed resistivity replaces the current model.

There will be a (small) relative error made by using the Frechet derivatives instead of elcardinal when estimating the data of the perturbed model. This error will mainly depend on the size of the perturbation and the depth of where the perturbation takes place (see figure 3.5). If the error is large, then we risk that moves accepted on the basis of the Frechet estimates would not have been accepted if we used elcardinal to calculate the data, and vice versa. Therefore, the accepted model will not be a good sample of the a posteriori probability distribution.

However,because the step lengths in each layer are updated according to the acceptance percentage,this will not create a huge problem. Shallow layers should give stronger responses and therefore these layers will eventually end up having low maximum step lengths. The problem that the relative error is too high will mostly occur in the beginning of the Monte Carlo simulation when the maximum step lengths have not yet stabilized.

To avoid saving models with Frechet estimated data that differs significantly from elcardinal-calculated data, a simple error-test is included in the script. For every nth accepted perturbation on a model, the relative error in the misfit made by using the Frechet estimates is calculated. If this error is higher than 10 %, then all of the perturbations on this model are erased, and the perturbation loop starts over beginning with the previous saved model. If the relative error is small, the script continues accepting the models as samples.

When all parameters have varied once, but before the model is saved ,the error test is performed again to ensure that the new model represents a sample of the a posteriori probability distribution. Then, if the stopping criteria are not met, the perturbation loop on each layer starts again, but this time the perturbations are made on this newly saved model.

The source codes, (the multilayered Monte Carlo script and the code that ensures the parallel computing), can be found in appendix B.3 and B.2.

3.4 Answering questions

A multitude of models giving data that sufficiently fit the data obtained from the reference model will exist. It is therefore important to evaluate all of these models. Because of the multidimensionality of the problem, the joint probability distribution is impractical or even impossible to visualize and we therefore plot the marginal a posteriori probability distribution instead. This can be done by counting the number of models with resistivity values within small intervals for each layer. Then histograms with the number of models on the y-axis and resistivity intervals at the x-axis can be created for each of these layers. If we want to look at the marginal

probability distribution of models at all layers in conjunction, we can use a color scale to represent the number of models and plot the resistivity and depth on the x- and y-axis respectively.

If the marginal probability distributions are Gaussian, then we could plot the mean model with the standard deviations in each layer. We will then be able to see that the resistivity in the layers that are best resolved will have smaller standard deviations. [3]

However, this will not give any information about how the combination of resistivities is distributed (the joint probability distribution). Therefore, another question that could be asked is how correlated the resistivity is for different layers. *Does the resistivity in one layer influence the resistivities in another layer?* The correlation matrix which is the normalized covariance matrix

$$c(i,j) = \frac{Cov(i,j)}{\sigma(i)\sigma(j)} = \frac{< (\rho_i - \overline{\rho_i})(\rho_j - \overline{\rho_j}) >}{\sigma(i)\sigma(j)} \tag{3.5}$$

can therefore be useful to have a look at. Here i and j denotes the layer index, ρ represents the resistivity and σ the standard deviation of the marginal resistivity distribution. Note that for $i = j$, the covariance $Cov(i,i)$ equals the variance in layer i $< (\rho_i - \overline{\rho_i})^2 >= \sigma_i^2$ and therefore the diagonal elements of the correlation matrix should be 1. The largest correlations will presumably be the correlation between adjacent layers. Deep layers should be (close to) uncorrelated due to the data alone because deep layers are expected to only weakly affect the data. The correlation between adjacent layers as well as the correlation of one layer with all the other layers could give valuable information. If, however, the a posterior probability distribution is far from Gaussian, calculating the covariance is meaningless.

Two additional quantities that could be interesting to calculate, is the average resistivity in the models and the first order moment of the a posteriori probability distribution. These two quantities may be calculated by

$$\overline{\rho} = \frac{\int_z \rho(z)dz}{\int_z dz} \approx \frac{\sum_i \rho_{i+1}(z_{i+1} - z_i)}{\sum_i (z_{i+1} - z_i)} \tag{3.6}$$

and

$$\mu_1 = \frac{\int_z z\rho(z)dz}{\int_z \rho(z)dz} \approx \frac{\sum_i \rho_{i+1}(z_{i+1} - z_i)^2}{2\sum_i \rho_{i+1}(z_{i+1} - z_i)}. \tag{3.7}$$

The first order moment of the probability distribution for each model is here divided by $\int_z \rho(z)dz$. This quantity gives the depth position of the " center of resistivity ".

One could also use the results to test a hypothesis on the resistivity profile. A question that could be asked is *How probable is it that there exist layers with resistivity higher than 20 Ωm in the range 1000-1500 m below the sea*

[3] We could also plot the median with mean deviations which is better if the marginal a posteriori probability distributions are strongly skewed

floor?. This question could be answered by counting the number of models sampled from the a posteriori probability distribution that fulfill the requirement and divide this number by the total number of sampled models. Although this question seems natural to ask at first, the models where the resistivity is 15 Ωm in two adjacent layers (at depth 1000-1500 m) will not be counted, but these cases are just as interesting.If we discretize our model space so that there is 100m between each layer, then a model with two adjacent layers each with a resistivity of $15\Omega m$ could be the result of a actual 100 m thick reservoir of $30\Omega m$ with its center at the layer boundary between the two neighboring layers.

Instead, one can ask *What is the probability that total resistivity in the range 1000-1500 m below the sea floor is greater than some constant?* Then, the problem can be written

$$P(\int_{1000}^{1500} \rho(z)dz \approx \sum_{i=10}^{15} d_i\rho_i > \text{constant}) \tag{3.8}$$

Chapter 4

Synthetic examples

To investigate properties of the a posteriori probability distribution, it is useful to test the method on synthetic reference models. In this chapter we look at three synthetic reference models, all of which consists of highly resistive, infinite, horizontal layers in a uniform background. The two first reference models both contain a single highly resistive layer, but at different depths, and the third reference model consists of two high resistive layers.

4.1 A priori information

Because the solution to an inversion problem is non-unique and many of the solutions may give models that are not physically realistic, it is useful to introduce constraints that will exclude non physical models. This is included as the a priori information. However, the a priori information should be limited so that the data of a measurement still significantly influences the a posteriori information.

One type of a priori information that will be used here is that unsmooth models are non-physical. We would therefore want to exclude such models. One way of achieving this is to introduce a regularization term in the misfit given by equation 2.18. The modified misfit for a model $\mathbf{m}$ then reads

$$S(\mathbf{m}) + R(\mathbf{m}) = \sum_{\text{offset}} |E_x^{\text{true}} - E_x^{\text{synt}}|^2 w_x + R(\mathbf{m}) \tag{4.1}$$

where the weights w_x are given by equation 2.19 and $R(\mathbf{m})$ is the regularization term which will here be given by

$$R(\mathbf{m}) = \int_z \left(\frac{\delta\sigma'}{\delta z}\right)^2 dz \approx \sum_{i=0}^{N} \left(\frac{\sigma'_{i+1} - \sigma'_i}{z_{i+1} - z_i}\right)^2. \tag{4.2}$$

Here σ' is the log10 value of the conductivity and z the depth beneath the sea floor, N is the number of layers, σ'_i the log10 value of the conductivity in layer number i, and z_i is the depth of layer number i .

Including the noise level which is usually in the range, $10^{-13}\frac{V}{Am^2} < \eta < 10^{-16}\frac{V}{Am^2}$, means that we must also add noise to the data obtained for the synthetic model. [1]

We can then write equation 2.11

$$\sigma(\mathbf{m}) = \rho \exp(-S(\mathbf{m}) - R(\mathbf{m})) \tag{4.3}$$

$$= \rho \exp(-R(\mathbf{m})) \exp(-S(\mathbf{m})) \tag{4.4}$$

$$= \rho_{new} \exp(-S(\mathbf{m})). \tag{4.5}$$

Note that here σ denotes the a posteriori probability density and ρ is the a priori probability distribution, not the conductivity and resistivity.

Other regularization formulas than the one given in equation 4.2 can also be used. One possibility is to include some depth dependence, another is to include regularization based on the gradient between resistivities and not on the square of the gradient.

Introducing the regularization term in the a priori sampling instead of including it in the likelihood function, saves a little computation time. Sampling the new a priori probability distribution can be done by using the metropolis rule to modify a uniform sampling according to the " likelihood function due to regularization " given by

$$L_{reg}(\mathbf{m}) = exp(-R(\mathbf{m})). \tag{4.6}$$

One sample of the uniform distribution that is accepted by the metropolis rule,will be a sample of the a priori probability distribution. Then the algorithm moves on to the metropolis rule based on the data misfit alone. If however the move is rejected based on the a priori information, then this is not a sample of the a priori probability distribution and we may throw away the model without checking the misfit due to data. Therefore we only need to calculate the data misfit (calling elcardinal) when a model has been accepted based on the a priori information.

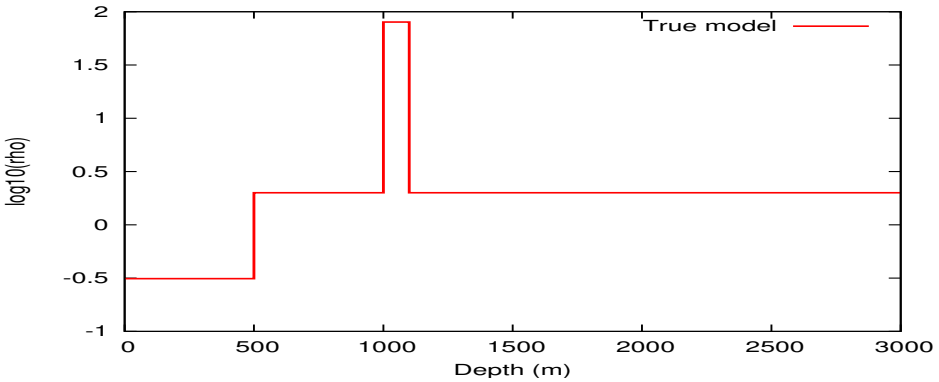
Because there is no need to calculate the electric fields from the models (they are only included in the $-S(\mathbf{m})$ term of the misfit), sampling the a priori distribution is much quicker than sampling the a posteriori distribution.

4.2 Description of the first reference model

In figure 4.1 the first reference model is shown. The sea floor is 500 m below the surface, and the high resistive target ranges from 1000 to 1100 m below the surface. The background resistivity is $2.0\Omega m$ and the target has a resistivity of $80\Omega m$.

[1] Adding noise to the data is easily done by using a program, ggAddNoise, developed by EMGS.The noise added is random and uniformly distributed between -noise level and noise level for each receiver

Figure 4.1 The first synthetic reference model



A high resistive layer is found at 1000-1100 m below the surface and the water depth is 500.0 m. The resistivity of this target is 80.0Ωm and the background resistivity is set to 2.0Ωm

4.2.1 Visualizing the a priori information

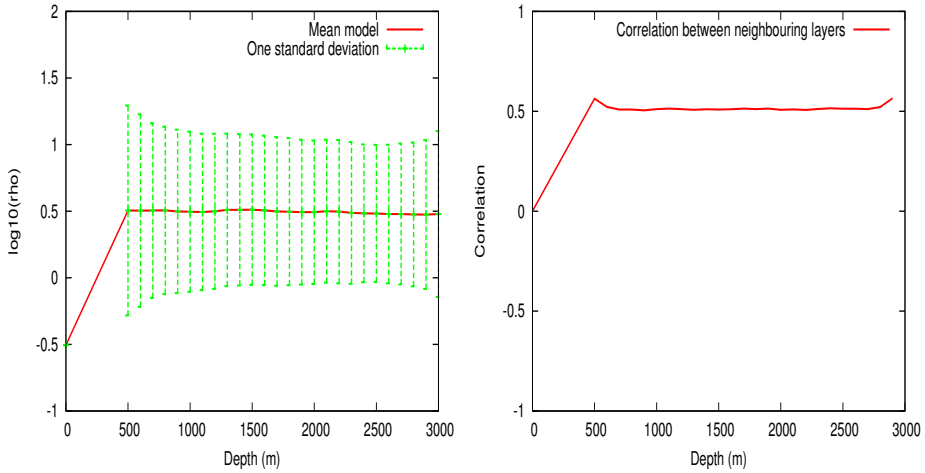
There are two reasons for investigating the a priori information. One is to verify that the algorithm is working correctly, the other is to understand what information we are putting into the problem.

In figure 4.2 we see the obtained a priori information for the synthetic reference model shown in figure 4.1.Each model consists of equally thick layers beneath the sea floor, with the exception of the deepest layer. This layer ranges to infinity. Here, the thickness of the layers is 100 m and the number of layers is 26. Thus, the deepest layer takes on resistivities representing the effective resistivity from 3000m below the surface to infinity.

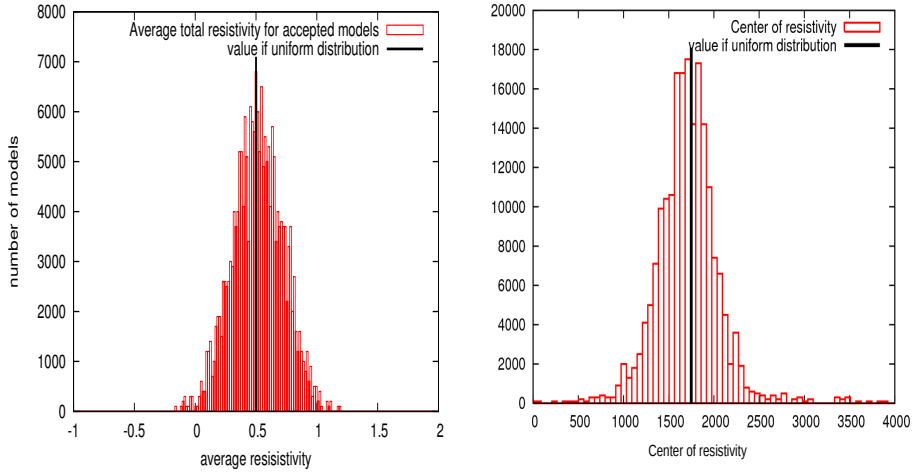
The sampling was done with a mapping $\rho^{'} = log10(\rho)$ and samples were chosen uniformly between $\rho^{'}_{max}$ and $\rho^{'}_{min}$. Because the upper and lower limit in this case was set to $1.0\Omega m < \rho < 100.0\Omega m$, the samples are chosen uniformly from -1 to 2. This should give an average resistivity of $rho^{'} = 0.5$. The plot in figure 4.2a shows the mean sampled resistivity at each layer with the standard deviations.The number of models sampled from the a priori probability distribution is 200 000. We see that the standard deviations are larger at the top and bottom layers. This artifact comes from the regularization term; The regularization for the top layer is defined by the difference in resistivities between this layer and the layer below it. For the other layers, the regularization is defined by the difference in resistivity with both the layers above and below it. This allows the top layer to vary more freely. The same argument goes for the bottom layer.

Figure 4.2b shows the correlation between adjacent layers. The correlation is highly positive,meaning that if the resistivity in one layer is high,

Figure 4.2 The a priori information



a) Mean model and std. deviations b) Neighboring correlations



c) Distribution of average resistivities d) Distribution of center of mass

a)In this figure the mean model is plotted together with the standard deviation in each layer. The a posteriori probability distributions are Gaussian,and therefore this plot gives the intended information b)The plot shows the correlation between adjacent layers. This correlation is an effect of the regularization term alone, as the data misfit is not included in the a priori sampling. c) This plot shows the a priori distribution of the average resistivity between the sea floor and 3000 m below the sea floor. The upper and lower limit for the resistivity is 100.0Ωm and 1.0 Ωm respectively, giving upper and lower limits for $log10(\rho)$ 2 and -1. Sampling uniformly from -1 to 2 should give an average of 0.5. d) Here we see the distribution of the first order moment divided by the total resistivity. This is the center of resistivity. If the resistivity is uniformly distributed between the layers then the value should be 1750.0

it is more likely that the value in an adjacent layer is also high. This is as expected since only the regularization determines the a priori distribution of models. Higher correlations at the top and bottom layers have the same explanation as for the standard deviations in figure 4.2a.

Figures 4.2c and d show the quantities given in equation 4.7 and equation 4.8.

$$\overline{\rho} = \frac{\int_z \rho(z)dz}{\int_z dz} \approx \frac{\sum_i \rho_{i+1}(z_{i+1} - z_i)}{\sum_i (z_{i+1} - z_i)} \tag{4.7}$$

$$\mu_1 = \frac{\int_z z\rho(z)dz}{\int_z \rho(z)dz} \approx \frac{\sum_i \rho_{i+1}(z_{i+1} - z_i)^2}{2\sum_i \rho_{i+1}(z_{i+1} - z_i)}. \tag{4.8}$$

The average resistivity for each layer when sampled uniformly between -1 and 2 should be $\rho^{'} = 0.5$. Figure 4.2c shows histograms of the number of sampled models versus the average resistivity over all layers in a model. As expected the most represented value is 0.5. For uniformly distributed resistivity (all layers have the same resistivity) the expression in equation 3.7 reduces to

$$\frac{maxdepth + waterdepth}{2}$$

which in this case gives a value of 1750.0. We see that the models with uniform resistivity is the most represented (figure 4.2d, and that the distribution has a Gaussian shape.

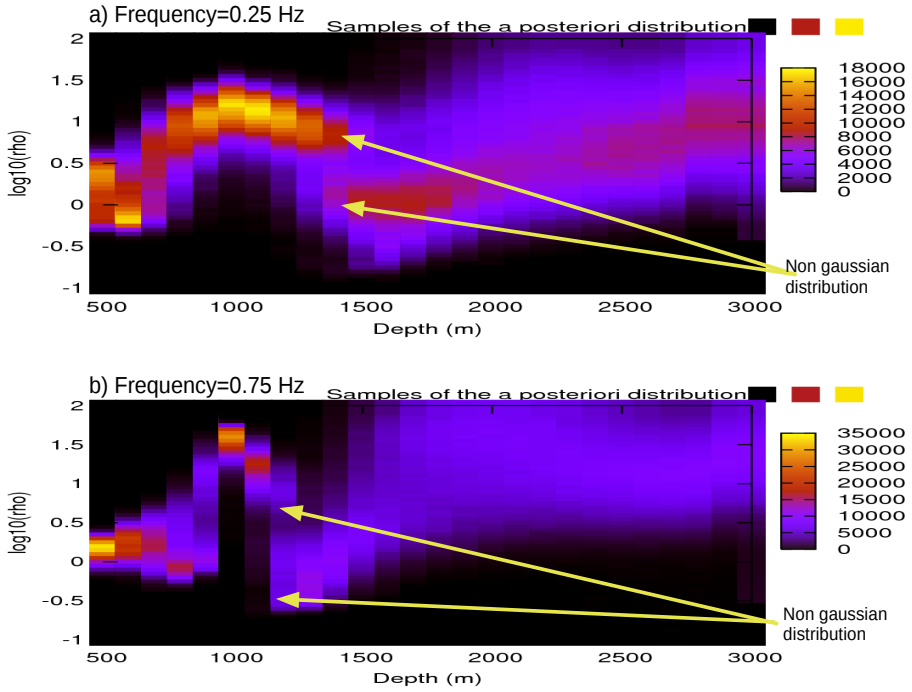
4.2.2 A posteriori information

The results in this section were obtained for several source frequency combinations and the number of models that were skipped at the beginning of the Monte Carlo search was 500. In total, the number of saved models were 200 000.

Figure 4.3a and b shows the marginal a posteriori probability distribution for the resistivity at depths between 500 m and 3000m below the surface (remember that the waterdepth is 500 m) when using the source frequencies 0.25 Hz and 0.75 Hz respectively. First let's discuss the similarities between these two results.

In both figures 4.3 a and b we see that the target layer at 1000-1100 m is better resolved (brighter colored peak) than most of the other layers. We also notice that the first couple of layers are very well resolved. However, most of the sampled resistivity values just above and below the target are lower than the true value (the true value of the background is $log10(\rho = 2) \approx 0.30$).Another similarity between the two results is that for the deeper layers, the marginal probability distribution is more smeared than for the top layers. This shows that the data contain less information about the deeper layers than the top layers.

Figure 4.3 Marginal a posteriori probability distribution



Distribution of models of the a posteriori probability distribution when separately using source frequencies 0.25 Hz and 0.75 Hz

If we look at the layers pointed out by the arrows in figure 4.3 a and b we see that the a posteriori distribution has two probability peaks. For such distributions the average resistivity in this layer lies between the two peaks and is not probable at all. Therefore, plotting the average model and its standard deviations as for the a priori distribution in figure 4.2 is misleading.Also, computing the covariances in these layers is meaningless.

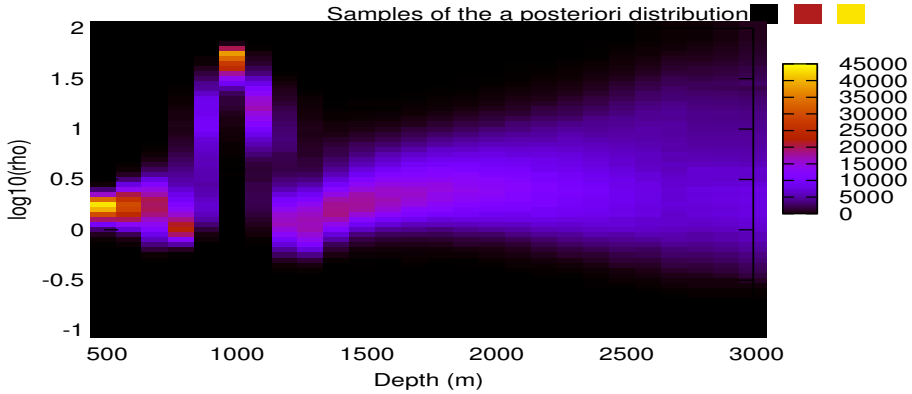
It is seen from figures 4.3a and b that the average resistivity in the bottom layers is much higher than the true value in the reference model. We also see this effect in figure A.1a and b. The equations 3.6 and 3.7 were used to create the plots showing the average resistivity and the depth distribution of the resistivity for the models sampled from the a posteriori probability distribution. We also see from these plots that too much high resistivity is distributed too deep.

Moving on to the differences in the two results, we see that using a source frequency of 0.25 Hz results in worse resolution of the target layer than for a source frequency of 0.75 Hz. But this does not mean that using

0.75 Hz is better. Figure 4.3 also reveals that deeper layers (1500 m - 2500 m) are slightly better resolved when using a source frequency of 0.25 Hz compared to using a 0.75 Hz source frequency. Therefore we would expect that using both frequencies in the Monte Carlo simulation will give better results than either frequency alone.

4.2.3 Using multiple frequencies

Figure 4.4 Marginal a posteriori probability distribution



Distribution of models from the a posteriori probability distribution when jointly using frequency 0.25 Hz and 0.75 Hz

In figure 4.4 the resulting marginal a posteriori probability distribution when using both source frequencies is displayed. We see many of the same features here as in figure 4.3, but the overall resolution has increased dramatically. The target layer have been at least as well resolved as for the 0.75 Hz source frequency case,and the deeper layers have been much better resolved than for either frequency alone.Even though the result is much better, it is not perfect. The total resistivity and how the resistivity is distributed are much closer to values calculated directly from the reference model. Plots showing this can be found in figure A.1 c.

So far we have seen that using both source frequencies, 0.25 Hz and 0.75 Hz, results in a sampling of models that are closer to the reference model than when using each source frequency alone.

> Is there a limit to the resolution due to increasing the number of source frequencies?

Pursuing this question, another three different frequency combinations were used to create samples from the a posteriori probability distributions.

These were $f_1 = 0.25\text{Hz}, 1.0\text{Hz}$, $f_2 = 0.25\text{Hz}, 0.5\text{Hz}, 0.75\text{Hz}, 1.0\text{Hz}$ and $f_3 = 0.25\text{Hz}, 1.0\text{Hz}, 3.0\text{Hz}$

The resulting a posteriori probability distributions are shown in figure A.2. Here we see that for frequency combinations f_2 and f_3 the resolution is better than for frequency combination f_1. This tells us that both increasing the frequency density and the upper frequency limit results in better resolution. However, the improvement is not nearly as strong as it was when going from one source frequency to two (figures 4.3 and 4.4).

One way of comparing the results in more detail is by answering the question (mentioned in section 3.4): *What is the probability that a layer at a given depth has got a resistivity value within a specific interval?*.

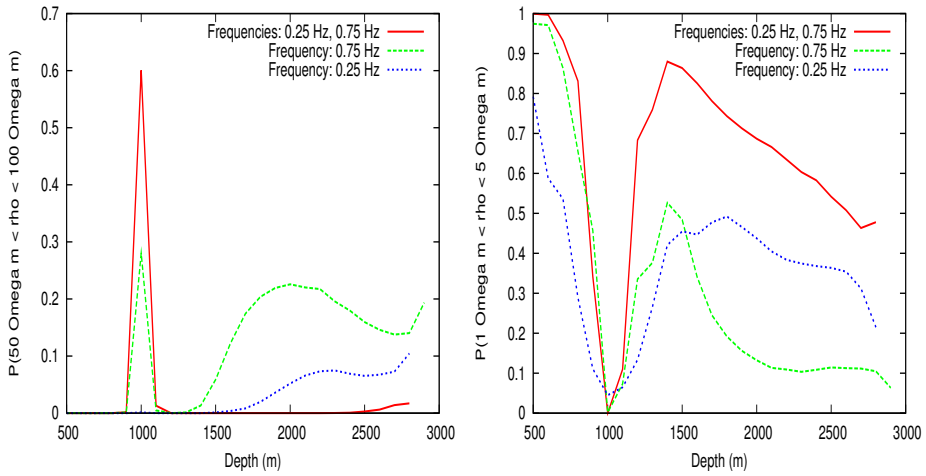
This probability is found by counting the number of models from the a posteriori distribution that obey the given criteria, and divide this number by the total number of models. (The source code for this calculation can be found in appendix B.5).

The plots in figure 4.5 shows the probability that a layer has a resistivity in a specific interval vs. the depth of the layer. To produce the results in figure 4.5 a and b, the a posteriori probability distribution of models when using the source frequencies 0.25 Hz or 0.75Hz alone and the combination of them were used. In figure 4.5 a the probability that a layer has a resistivity value in the interval $50\Omega m < \rho < 100\Omega m$ is plotted as a function of depth. The point on the x-axis, f.ex at 1000 m, is the upper boundary of the layer. Remembering that the layer thickness is 100 m, this means that the better the resolution, the higher the probability in the layer with upper boundary at 1000m, and the lower the probability in the other layers.

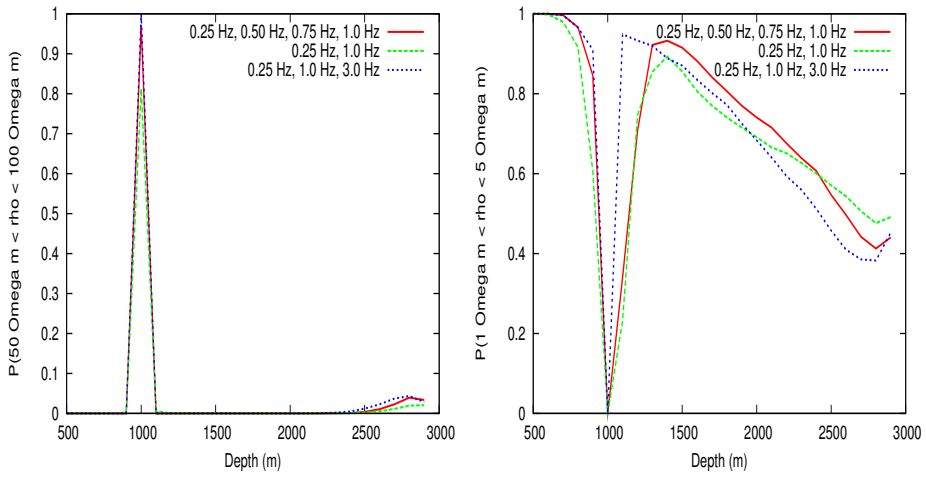
We clearly see the improvement when jointly using the frequencies instead of using either one alone. When using only the 0.25 Hz source frequency,the probability that the layer at 1000 m - 1100 m has a resistivity over $50\Omega m$ is zero, but around 8% in the layers deeper than 2000 m. For the 0.75 Hz source frequency there is about 28% probable that the layer at 1000m - 1100 m has a resistivity between $50\Omega m$ and $100\Omega m$. However, the probability in the deeper layers is also much higher than for the 0.25 Hz case. When using both frequencies, the probability that the background layers have resistivity values above $50\Omega m$ is close to zero, and for the target layer the probability is about 60 %.

In figure 4.5 b the probability that the layers have a resistivity value in the range $1.0\Omega m < \rho < 5.0\Omega m$ as a function of depth is displayed. This plot gives a better insight on how well the background layers have been resolved. We see that when using the 0.75 Hz frequency there is only a 5 - 25 % chance that the layers below 1800.0 m have a resistivity between 1 Ωm and 5 Ωm, while for the 0.25 Hz case, the probability is between 20 - 50 %. In the top layers, the opposite is the case. The deeper layers are clearly better resolved for the lower frequency and vice versa. When jointly using both frequencies, the background resistivity resolution has been signif-

Figure 4.5 The probability that a model sampled from the a posteriori probability distribution has a resistivity in a specific interval at depth



a) $P(5000.0 < 100m \cdot \rho < 10000.0)$ b) $P(100.0 < 100m \cdot \rho < 500.0)$



a) $P(5000.0 < 100m \cdot \rho < 10000.0)$ b) $P(100.0 < 100m \cdot \rho < 500.0)$

icantly improved both for deep and shallow layers.

Moving on to the plots in figure 4.5 b and c where the a posteriori probability distributions for the frequency combinations f_1,f_2 and f_3 have been used, we see that the difference in the resolution is much smaller than for figures 4.5 a and b. There is a slight improvement when using all of the frequencies 0.25 Hz, 0.50 Hz, 0.75 Hz and 1.0 Hz compared with using only 0.25 Hz and 1.0 Hz.There is also an improvement in the top layers when using the frequency combination 0.25 Hz,1.0 Hz and 3.0 Hz instead of 0.25 Hz and 1.0 Hz. These results show that there is little to gain by increasing the frequency density and frequency range beyond some reasonable limit,

which is consistent with the conclusion in reference [22].

4.2.4 Using another start model

In all the previous Monte Carlo simulations the reference model was used as start model. The a posteriori probability distribution should be independent of the start model if we wait long enough with saving the models. Starting with the reference model we know that we are close to the global minimum [2], and samples representing the a posteriori distribution are picked from the beginning.

To verify that the start model does not have much influence on the a posteriori probability distribution, the Monte Carlo simulation when jointly using the source frequencies 0.25 Hz and 0.75 Hz was rerun, starting with a uniform start model. The resistivity was $2.0\Omega m$ at all depths, and the number of layers were increased so that the bottom layer took on resistivity values representing the effective resistivity at 3500m below the sea floor to infinity. Everything else was kept the same.

By skipping the first 500 models on every process [3] before saving models the a posteriori probability distribution figure A.3 was produced. The number of models from the a posteriori probability distribution was 200 000.

We see from figure A.3. that, as expected, the start model did not significantly influence the result.

4.3 The second synthetic reference model

In section 4.2 the target layer was placed at a shallow depth beneath the sea floor and is therefore considered an easy target to resolve. It is usual that the reservoirs are found much deeper. The depth dependence of the resolution seen in section 4.2, indicates that it is more difficult to detect deeper high resistive targets.

The next reference model consist of a single high resistive layer at 2500m-2600 m below the surface. The background resistivity is $2.0\Omega m$, and the water depth 500 m. The regularization term in unchanged (see equation 4.2), therefore the a priori information is the same as in section 4.2.1.

The focus in this section will be on how the a posteriori probability distribution changes due to different noise levels and data uncertainty. The results are obtained by jointly using the frequencies 0.25 Hz and 1.0Hz.

[2] We may not be in the global minimum because of regularization. If there was no a priori information in addition to the uniform sampling of models ,the reference model would be in the global a posteriori probability minimum

[3] 100 processes were used so that the total number of skipped models was 50 000

4.3.1 A posteriori information

The marginal a posteriori probability distributions are shown in figure 4.6. Looking at figure 4.6 a where the added noise,η, is $|\eta| < 10^{-13}V/Am^2$ and assumed data uncertainty is 10 %, we see that models sampled from the a posteriori probability distribution have a slight tendency to contain high resistive layers around 2000 m below the surface. However, the probability distribution is strongly smeared, especially for the deep layers (> 1500 m). Although decreasing the data uncertainty does not improve the sharpness of the a posteriori probability distribution (figure 4.6c), the depth at which the models are most likely to have a high resistive layer is more accurate: High resistive layers are most likely to be found around 2500 m below the surface.

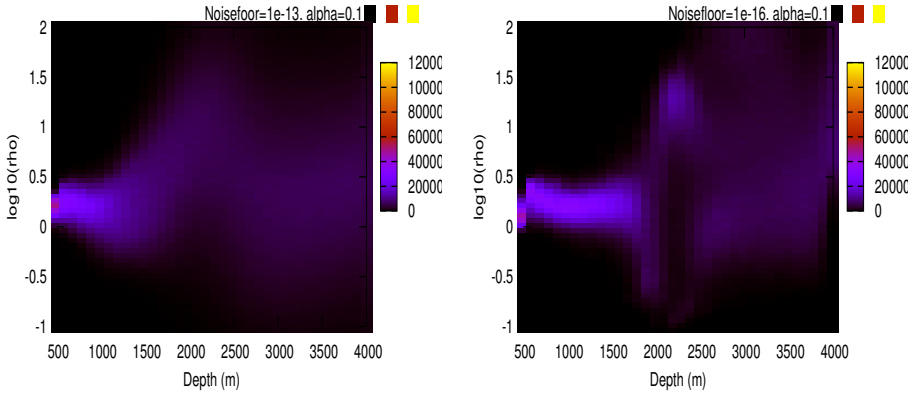
With lower background noise,$|\eta| < 10^{-16}V/Am^2$, (figure 4.6 b and d) the probability distributions are less smeared, and the existence of a high resistive layer is better seen. Here, the result of improving the data uncertainty is clearly seen in the a posteriori probability distributions. In figure 4.6 b the a posteriori probability distribution is drastically smeared below 2500 m, whereas the probability distribution in figure 4.6 d the probability distribution in the deeper layers is much sharper.Both of these figures show that the the most probable models have a high resistive layer at around 2300 m below the surface.

In table 4.1 the probabilities that the average resistivity is more than $10\Omega m$ at different depths intervals is displayed for each of the results in figure 4.6. All depth intervals are here 500m thick, but the placement varies from 1500m-2000 m to 2500-3000m below the surface.

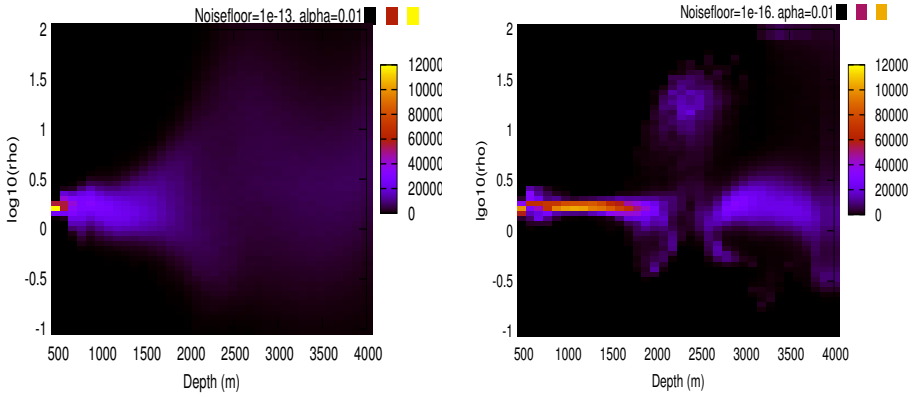
From this table we see that it is only for the lowest noise level and data uncertainty that the highest probability of high average resistivity is placed at the correct depth (blue boxes). For both cases where 10 % data uncertainty was assumed, the probability that the average resistivity is $10\Omega m$ or more, is slightly higher at 2000 - 2500 m than for 2200 - 2700 m below the surface. However, the number of models with high resistive layers is significantly higher for low noise level both in the interval 2000-2500 m and 2200-2700 m. For noise floor $10^{-13}V/Am^2$ and data uncertainty of 1% the depth which has the highest probability of consiting of high resistive layer is too deep (2500-3000 m).

In both table 4.1 and figure 4.6 we see that it is significantly harder to detect high resistive layers that are buried deep. Detecting such targets require very low background noise and data uncertainty.

Figure 4.6 A posteriori probability distributions



a) $|\eta| < 10^{-13} V/Am^2$. α=10% b) $|\eta| < 10^{-16} V/Am^2$. α= 10%



c)$|\eta| < 1^{-13} V/Am^2$. α=1% d) $|\eta| < 10^{-16} V/Am^2$. α= 1%

In a) and b) the data uncertainty is assumed to be 10%. The noise added to the synthetic data,η , is $|\eta| < 10^{-13} V/Am^2$ and $|\eta| < 10^{-16} V/Am^2$ for a) and b) respectively. In c) and d) 1% data uncertainty is assumed and the noise added to the data is $|\eta| < 10^{-13} V/Am^2$ in c) and $|\eta| < 10^{-16} V/Am^2$ in d)

Table 4.1: The probability that a model sampled according to the a posteriori probability distribution has an average resistivity above $10\Omega m$ at different depth intervals

Result obtained for:	$P(\int_{1500}^{2000} z\rho(z)dz > 5000)$	$P(\int_{2000}^{2500} z\rho(z)dz > 5000)$	$P(\int_{2200}^{2700} z\rho(z)dz > 5000)$	$P(\int_{2500}^{3000} z\rho(z)dz > 5000)$
Noise: 1e-13 Data uncertainty: 10%	0.142205	0.56675	0.514395	0.3292
Noise: 1e-13 Data uncertainty: 1%	4.5 e-5	0.172285	0.34372	0.4832
Noise: 1e-16 Data uncertainty: 10%	0.0	0.78056	0.74161	0.331435
Noise: 1e-16 Data uncertainty: 1%	0.0	0.614995	0.843985	0.2949

4.4 The third synthetic reference model

The motivation for the next synthetic model is to see if it is possible to detect a high resistive layer below another high resistive layer. Also, the effect of the regularization term is investigated here.

The synthetic reference model consists of two high resistive layers. The shallow high resistive layer is at 1000 m below the sea floor and has a resistivity of $20.0\Omega m$ and the deep high resistive layer, with resistivity $100\Omega m$, is placed at 2000.0 m below the sea floor.

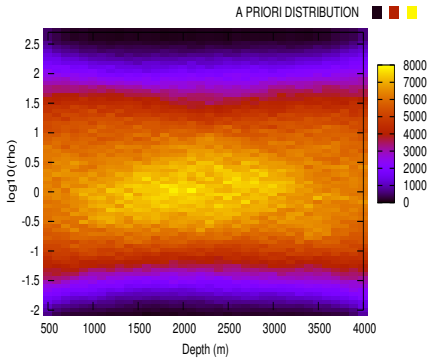
4.4.1 A priori information

The hard constraints imposed in the a priori information was the upper and lower limits for the resistivity $\rho_{max} = 500.0\Omega m$ and $\rho_{min} = 0.01\Omega m$.

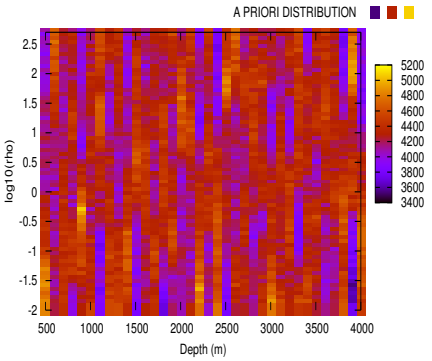
The a priori probability distributions and the a priori correlations between layers are shown in figure 4.7.The left plots (figure 4.7 a, c and e), are obtained when imposing smoothness on the a priori models by using the regularization term in equation 4.2 multiplied by a factor ten to increase its influence. The plots to the right (figure 4.7 b, d and f) are obtained when no a priori information apart from the hard constraints and the sampling described in section 3.2.3 is assumed.

We see that the marginal a priori distribution of models where smoothness is required in figure 4.7 a is Gaussian distributed for each layer.Without the smoothness requirement we see that the a priori distribution of models

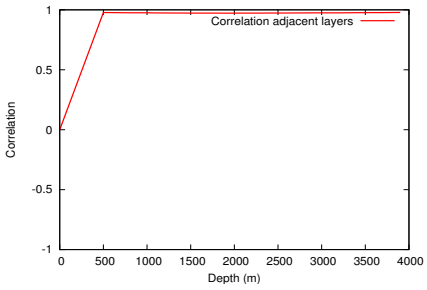
Figure 4.7 A priori information



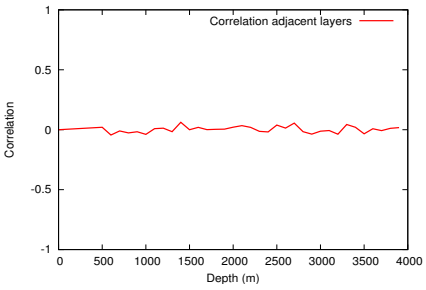
a) With strong regularization



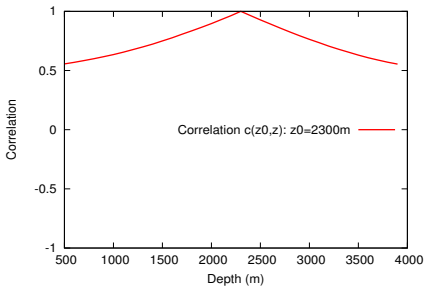
b) With no regularization



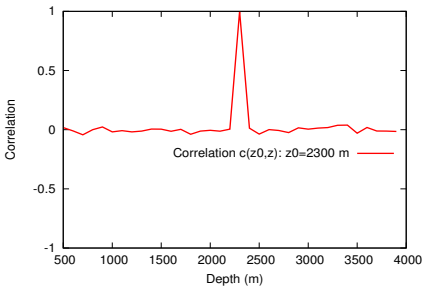
c) With strong regularization



d) With no regularization



e) With strong regularization



f) With no regularization

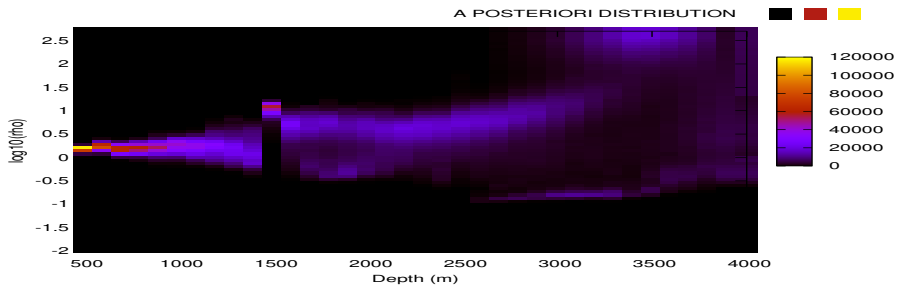
The a priori probability distributions when imposing different a priori information on the models. The plots a, c and e are obtained when using a strong smoothness constraint on the models. The plots in b, d and f are obtained when no a priori information in addition to to the sampling described in section 3.2.3 is imposed

is close to uniform (figure 4.7 b). The correlation between adjacent layers when including the strict smoothness requirement on models from the a priori probability distribution (figures 4.7 c) are positive and high.When no soft constraint is imposed on the models, then the correlation between adjacent layers fluctuates around zero (see figure 4.7 d).

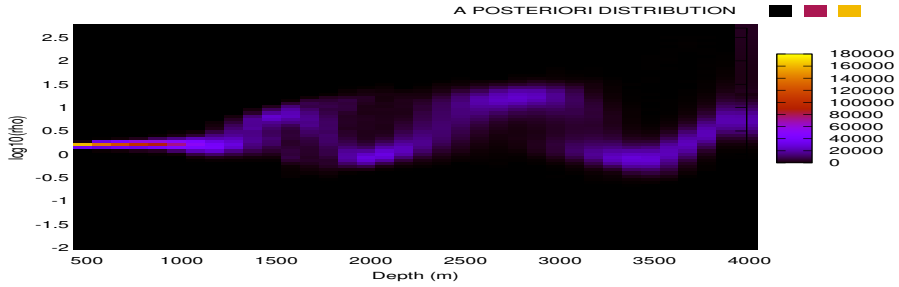
In figures 4.7 e and f the correlation function $c(i, j)$ given in equation 3.5 is displayed. The indices i and j represents the number of the layer, and here the layer $2300m < z_i < 2400m$ was chosen. From these plots we see that the resistivity in one layer is only correlated with itself when no a priori smoothness is assumed. When only smooth models are accepted as samples of the a priori probability distribution, then the resistivity in one layer is positively correlated with all the other layers. However, this correlation decreases as the distance to the other layers increases.

4.4.2 A posteriori information

Figure 4.8 Marginal a posteriori probability distributions



a) No regularization



b) Strict regularization

a) The result when assuming no a priori smoothness. b) The result when assuming that smooth models are much more correct

The a posteriori probability distributions when running the Monte Carlo

simulation both with and without the regularization term are found in figure 4.8. The results were obtained using an inline configuration. The noise level was $10e-15V/Am^2$ and the data uncertainty is assumed to be 2.5% of the amplitude. The results here were obtained using a half space model of $2.0\Omega m$ as start model. The number of models skipped at the beginning of the simulation for each job was 1000. The total number of models considered as samples of the a posteriori probability distribution was 200000.

It is clear from the distributions of models that including the regularization term has been very effective. A great number of non physical models have been excluded, and the result is a distribution of models that are much closer to the reference model.

Although the deepest layer is not resolved when no regularization is used, the total average resistivity beneath the sea floor is closer to the average resistivity of the reference model, than when smoothness is imposed. (See figure A.4.

The plots in figure 4.8 show the importance of the a priori information.Therefore, this information should be carefully chosen. If no regularization is used, then detecting the deepest target seems to be an almost impossible task. The shallow layer, however, is well resolved. In fact, the shallow layer is better resolved when only the data misfit is considered. This is an indication that the regularization should not be so strict in shallow layers, but that it is OK for the regularization to play an important part in deep layers. Using another regularization formula,

$$R(\mathbf{m}) = \sum_{i=0}^{N} \left(\frac{\sigma'_{i+1} - \sigma'_i}{z_{i+1} - z_i} \right)^2 \left(\frac{z_{i+1} - z_0}{5z_0} \right) \tag{4.9}$$

, where the misfit penalty due to large resistivity gaps between adjacent layers increases with depth, did not significantly improve the result. The resulting a posteriori distribution of models with the regularization in 4.9 can be found in figure A.5.

Chapter 5

Tests on real data

In this chapter we move on from using data obtained from synthetic to real reference models. The site under consideration is the Troll West Oil Province (TWOP). First, we briefly discuss some considerations that need to be taken into account when using real data instead of synthetic data. A short description of drilling results on TWOP will then be given, before the focus moves on to the results, both of start models found by the simulated annealing code in appendix B.6 and the a posteriori distribution of models for three different a priori assumptions.

5.1 From synthetic to real reference models

All synthetic reference models discussed so far were isotropic models meaning that the horizontal and vertical resistivities were forced equal in the Monte Carlo simulations. However, as mentioned in section 2.1.1, the horizontal and vertical resistivities in the seabed may not be equal in real world scenarios. This should be taken into consideration when using data from a real test site. By introducing horizontal resistivities as well as vertical resistivities, the number of model parameters is doubled and the model space is dramatically increased. Therefore, some a priori information due to the properties of anisotropy should be included.

One assumption that we make is that the horizontal resistivity is always less than the vertical resistivity [23]. Also the anisotropy ratio,

$$\lambda^2 = \frac{\rho_v}{\rho_h} \tag{5.1}$$

, is considered to be high if $\lambda^2 > 3.0$, and models with very high anisotropy ratio could be seen as non physical. Therefore, a soft constraint on the anisotropy ratio could also be included to ensure that models with anisotropy values above $\lambda^2 = 3.0$ are less probable than models with lower anisotropy ratio.

The noise that was present during the acquisition of real data, can be estimated directly. The noise values are then stored in a separate file which is read directly by the Monte Carlo script and used to calculate the weights in equation 2.19.

5.2 The Troll oil field

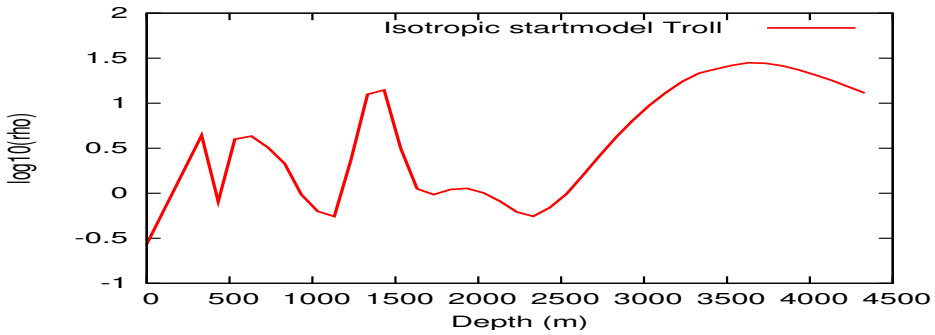
The Troll West, located in the North Sea, is a producing field with an approximate size of 10x2.5 km. The depth of the target is at around 1500 m below the surface and the water depth is around 300.0 m. In the following, we use data from a receiver positioned at the seafloor 332.0 m below the surface. The conductivity of the water was measured $\sigma = 3.69$ Sm^{-1}

5.3 Start models Troll

By running the simulated annealing script in appendix B.6 the start models for the Monte Carlo simulations were found. The smoothness requirement implemented by the regularization term in equation 4.2 was included in the simulated annealing search for both the isotropic and anisotropic start models (see figures 5.1 and 5.2).

In figure 5.1 the 1D inversion result when allowing only isotropic models is displayed. This start model clearly shows three resistivity peaks, one just below the sea floor, another at about 600 m below the surface and one with an even higher resistivity at around 1400 m below the surface. The background resistivity rises to very high values below 2500.0 m below the surface.

Figure 5.1 Isotropic start model found by 1D inversion on the Troll oil field

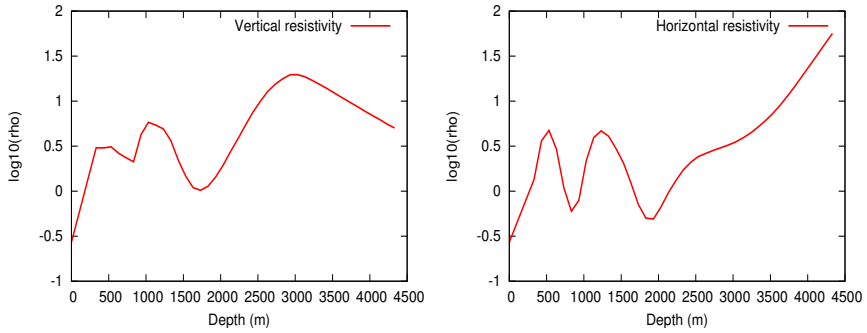


The start model found by the simulated annealing code in appendix B.6 when allowing only isotropic models. Smoothness was assumed by including the regularization term in equation 4.2.

The anisotropic start model in figure 5.2 was obtained when no constraint on the anisotropy was included. This was done because a hard constraint on the anisotropy could limit the search, and we are more likely to get stuck in a local misfit minimum.

The 1D inversion result here clearly shows two different profiles for the vertical and horizontal resistivities. Both the vertical and horizontal resistivity profiles differ from the isotropic start model. We see that the resistivity peaks at 400.0 and 600.0 m in the isotropic start model, has been combined into one peak for both the vertical and horizontal resistivity profiles. However, for the vertical resistivity profile, the gap between the two resistivity peaks (at around 500.0 m and 1300.0m below the surface) is much less defined than for the horizontal resistivity profile. The resistivity just below the second highly resistive peak is also lower in the horizontal case.

Figure 5.2 Anisotropic start model found by 1D inversion on the Troll oil field



a)Vertical resistivity b) Horizontal resistivity

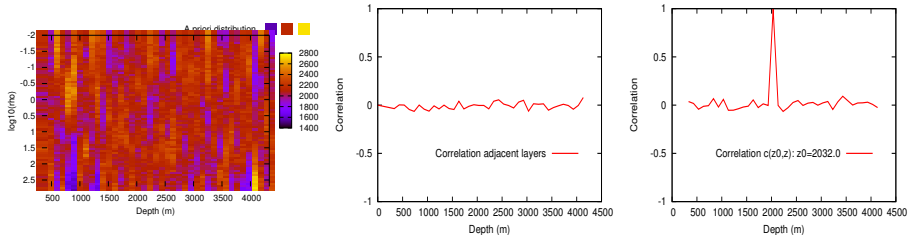
The start model is found by using the simulated annealing algorithm described in section 3.2.2. Here, no regularization due to the ratio between horizontal and vertical resistivity was included in the misfit function. The regularization used is the same as in equation 4.2.

We notice that the top high resistive peak and the layers below 3500.0 m below the surface have higher horizontal than vertical resistivities. However, at depths between 700.0 - 3500.0 m the horizontal resistivity is in general lower than the vertical.

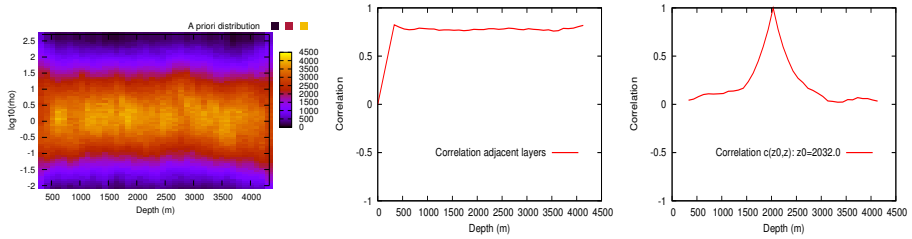
5.4 A priori information

In figures 5.3, 5.4, 5.5 and 5.6 the a priori information used in the following Monte Carlo simulations of both isotropic and anisotropic models on the Troll oil field is shown.

Figure 5.3 A priori information for isotropic models on the Troll oil field

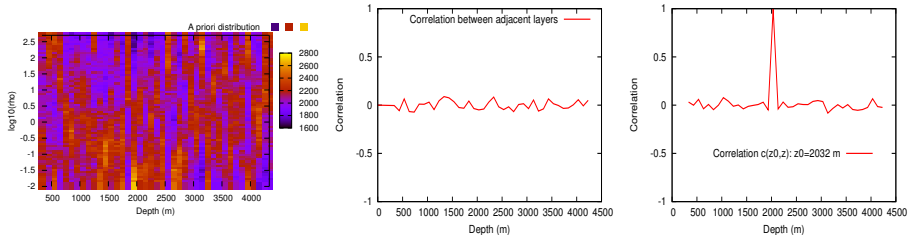


a) No regularization

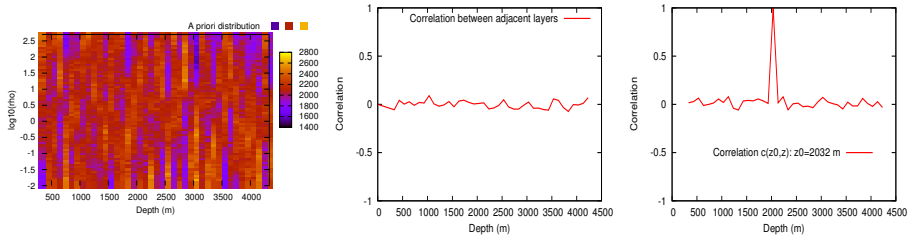


b) With regularization

Figure 5.4 A priori information for anisotropic models on Troll



a) Vertical resistivity

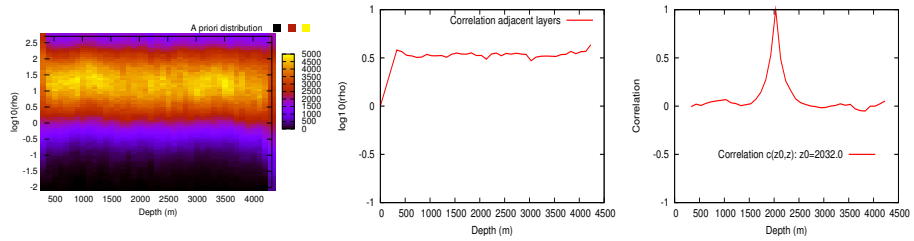


b) Horizontal resistivity

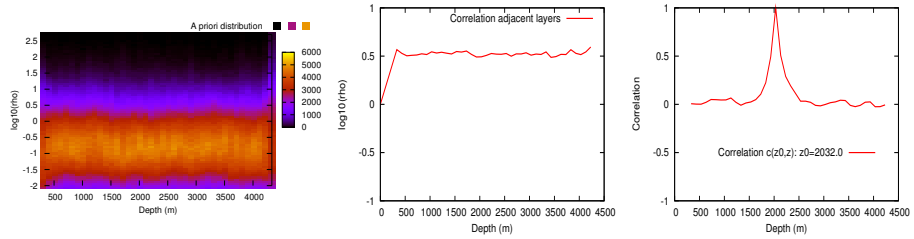
Here no a priori information in addition to the sampling described in section 3.2.3 is assumed

When assuming isotropic models only, we have used two different types of a priori information for our studies. First, we assumed no additional a priori information to the log10-sampling described in section 3.2.3. This

Figure 5.5 A priori information for anisotropic models on Troll



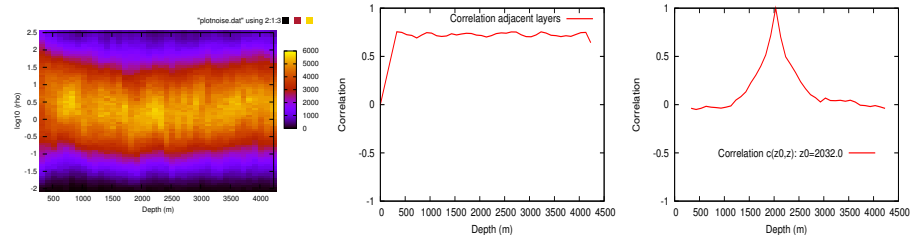
a) Vertical resistivity



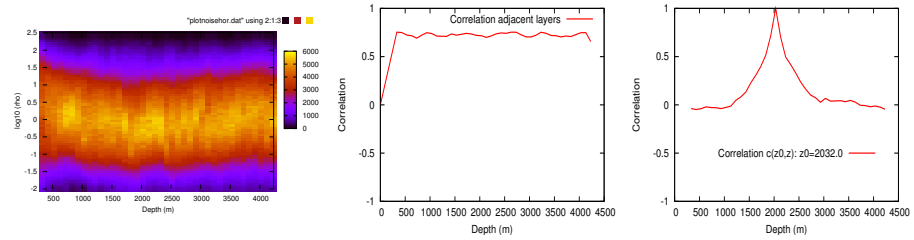
b) Horizontal resistivity

Here we have included a strong misfit penalty if $\rho_v < \rho_h$ in addition to the sampling described in section 3.2.3

Figure 5.6 A priori information for anisotropic models on Troll



a) Vertical resistivity



b) Horizontal resistivity

Here we have included a strict misfit penalty if $\rho_v < \rho_h$ and a weak penalty if the anisotropy ratio $\lambda^2 > 3.0$

ensured that the data strongly determine the a posteriori distribution of models. The a priori information can be found in figure 5.3 a.

We also run one Monte Carlo simulation imposing smoothness through the regularization term in equation 4.2 on the isotropic models. The a priori information in this case can be found in figure 5.3 b. The start model used to create the plots in figure 5.3 was the one shown in figure 5.1.

The a priori information when no smoothness was imposed on the models shows uniformly distributed resistivity values.The resistivities in each layer are more or less uncorrelated with the resistivities in the adjacent layers. The correlation of the resistivities in one layer with the resistivities in all the layers is 1 with itself and around zero with all the others. When including the regularization we see that the a priori distribution of models is Gaussian like in each layer, and that the resistivities in adjacent layers are positively correlated. The correlation between resistivities in one layer and all the other layers seems to decrease exponentially with the distance between the layers.

For the anisotropic models three different types of a priori information was used. In all three cases the start model was the one shown in figure 5.2.

In the first case no regularization was imposed (see figure 5.4). The only a priori information was the hard constraints on the upper and lower resistivity limits $\rho_{max} = 500\Omega m$ and $\rho_{min} = 0.01\Omega m$ and the a priori sampling described in section 3.2.3. We see that both the horizontal and vertical resistivity distributions consist of uniformly distributed (in the $log10(\rho)$ scale), uncorrelated models.

Figure 5.5 shows the a priori information when including both the smoothness requirement given by the regularization formula in equation 4.2 and including a strict misfit penalty if the horizontal resistivity in a layer is higher than the vertical resistivity. The effect is clearly seen in the a priori distribution of models. Here both the vertical and horizontal resistivities are Gaussian distributed in each layer, but the average vertical resistivity is around $\overline{\rho}_v = 10^{1.3}\Omega m$ while the average horizontal resistivity is about $\overline{\rho}_h = 10^{-0.3}\Omega m$. The correlation plots are strongly affected by the smoothness regularization.

The third a priori information that was used included the weak misfit penalty,

$$P(\mathbf{m}) = \sum_i \left(\rho'_{v,i} - \rho'_{h,i}\right)^2 \cdot k \tag{5.2}$$

where i denotes the layer number, v and h denotes the vertical and horizontal directions and

$$k = \begin{cases} 1.0 & \text{if} \frac{\rho_{v,i}}{\rho_{h,i}} > 3.0 \\ 0.0 & \text{otherwise} \end{cases} \tag{5.3}$$

together with the smoothness regularization and the strict misfit penalty on $\rho_v < \rho_h$. By doing this, we assume that very high anisotropy is physically unlikely. The a priori information is displayed in figure 5.6. We clearly see the effect of the additional constraint, the difference between the horizontal and vertical average resistivities is much less than in figure 5.5. Also we see from the correlation plots that these are almost identical. This means that the models that are accepted as samples of the a priori probability distribution are much closer to isotropic than for the previous case.

5.5 A posteriori information

The source frequencies used for acquisition during this marine CSEM inline survey on the Troll oil field was 0.25Hz, 0.75 Hz and 1.25 Hz. In the following results 2.5 % multiplicative data uncertainty was assumed. The total number of saved models sampled from the a posteriori probability distribution was 200 000.

5.5.1 Isotropic models

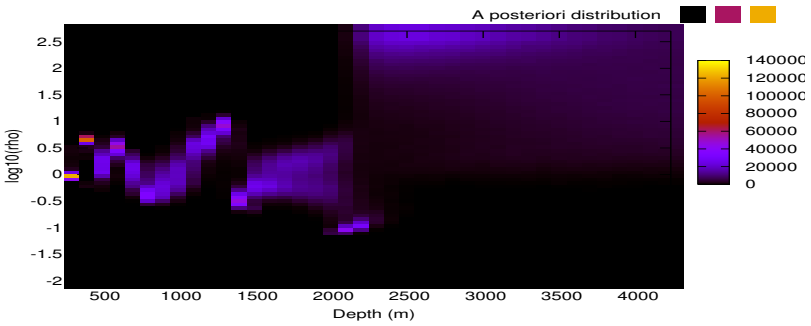
The a posteriori probability distributions when assuming isotropic models are shown in figure 5.7. With no a priori smoothness regularization we see that the start model found by simulated annealing fit fairly well with the a posteriori distribution of models (figure 5.7a). However, most samples of the a posteriori distribution have placed the high resistivities a few hundred meters shallower than in the start model (1100-1400 m below the surface). We also notice the sharp resistivity contrast at around 2100 m below the surface, where the resistivities suddenly become very high.

When the smoothness constraint was introduced, the a posteriori distribution of models (figure 5.7b) did not change much. The main difference between the figures 5.7 a and b is at 2000.0 to 3000.0 m below the surface. Here we see that the transmission from low to high resistivity is much smoother when including the smoothness regularization. Here too, most of the high resistivities are placed at 1100.0 -1400.0 m below the surface.

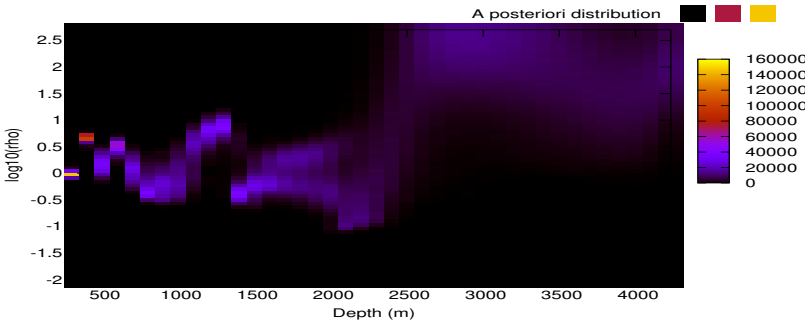
5.5.2 Anisotropic models

In figure 5.8 we see the a posteriori distribution of models when no a priori information on the anisotropy ratio or smoothness was assumed. Here the vertical resistivity (figure 5.8 a) shows a high resistivity area around 1100-1400 m below the surface. We see that the vertical resistivity's just above and below this highly resistive area have varied more freely in the log10 scale than at the high resistivity location. At depths deeper than 2000.0 m below the surface, very little information can be inferred about the resistivity profile.

Figure 5.7 Marginal a posteriori distribution of models



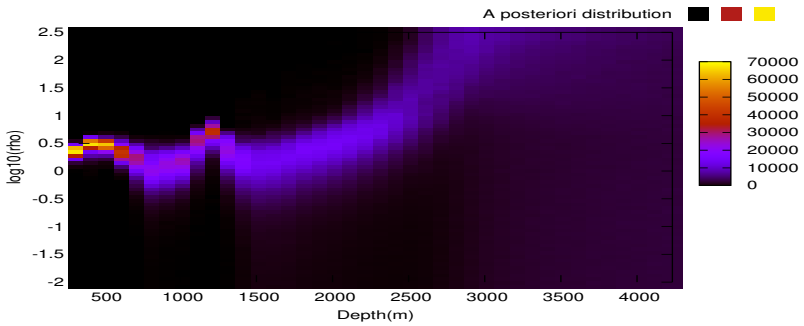
a) No regularization



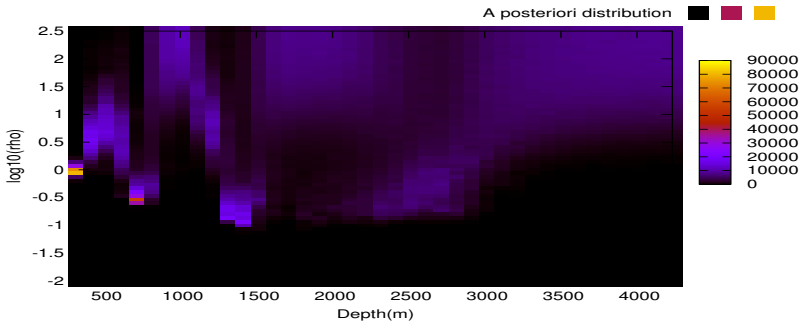
b) With regularization

The a posteriori distribution of models when no a priori information was assumed in addition to the uniform $log10(\rho)$ -sampling a). b) The a posteriori distribution of models when smoothness was assumed as prior information

Figure 5.8 Marginal a posteriori distribution of models



a)Vertical resistivity

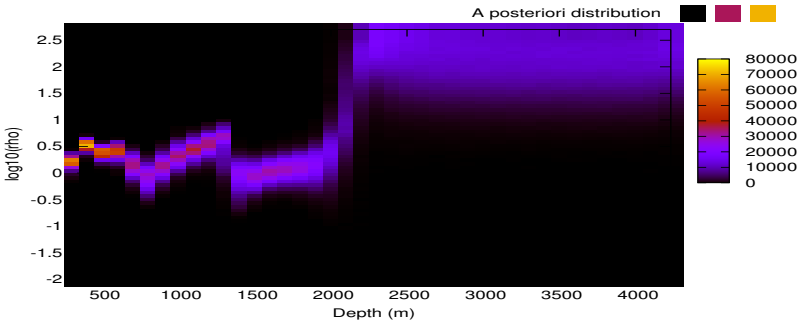


b)Horizontal resistivity

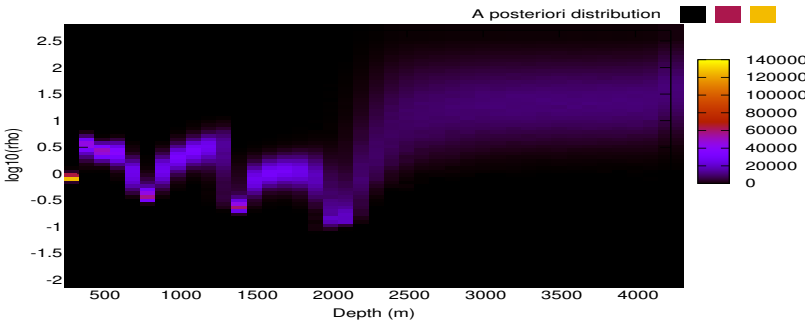
The a posteriori distribution of models when no a priori information was assumed. a) The a posteriori distribution of vertical resistivity at depth. b) The a posteriori distribution of horizontal resistivity at depth.

From the a posteriori distribution of horizontal resisitivities at depth (figure 5.8 b) however, we see that the distribution of models is more defined at high conductivity areas (at about 700.0-800.0m, 1300.0-1500.0 m below the surface). The top layer is also very well defined for the horizontal resistivities. At depths where the sampled models do not primarily consist of low horizontal resistivities, the distribution is highly smeared and little information can be inferred.

Figure 5.9 Marginal a posteriori distribution of models



a) Vertical resistivity



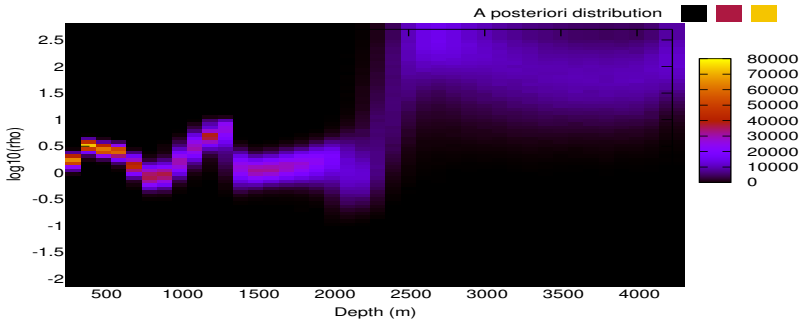
b) Horizontal resistivity

The a posteriori distribution of models when smoothness and a strict penalty if $\rho_{ver} < \rho_{hor}$ is assumed as a priori information. a) The a posteriori distribution of vertical resistivity at depth. b) The a posteriori distribution of horizontal resistivity at depth.

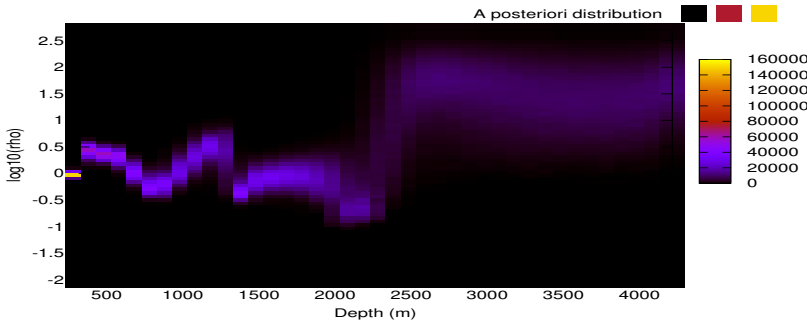
In figure 5.9 we see the a posteriori distribution of models when including a weak smoothness regularization and assuming that models where $\rho_{ver} > \rho_{hor}$ are physically unlikely. At depths up to about 2000 m,the horizontal resistivities follow the vertical resistivities closely. However, apart from the highly conductive areas, the resistivity profile is less defined.

At depths deeper than 2000 m below the surface, the a posteriori distribution of models for vertical and horizontal resistivities separates. Here,the distributions show high vertical resistivities than horizontal resistivities, but both distributions are highly smeared.

Figure 5.10 Marginal a posteriori distribution of models



a) Vertical resistivity



b) Horizontal resistivity

The a posteriori distribution of models when smoothness, the strict misfit penalty on $\rho_{ver} < \rho_{hor}$ and a weak misfit penalty on $\lambda^2 > 3.0$ is imposed. a) The a posteriori distribution of vertical resistivity at depth. b) The a posteriori distribution of horizontal resistivity at depth.

In figure 5.10 we see the a posteriori distribution of models when a misfit penalty on the anisotropy ratio $\lambda^2 > 3.0$ is included in addition to the a priori information for the previous case (figure 5.9). We see that this soft constraint has not influenced the result significantly. However, the horizontal resistivities follow the vertical a bit closer. At depths deeper than 2000.0 m below the surface, we see a small change from the result in figure 5.9. Here the horizontal and vertical resistivity distributions do not separate as much.

Chapter 6

Discussion

In this chapter we explain the physics describing some of the results obtained in the previous chapters. First we look at some general properties that were observed in both synthetic and real examples. Then we move on to more specific results, such as the effect of using multiple frequencies and different types of a priori information. Finally we discuss the results obtained on the Troll West Oil Province.

6.1 Some general properties of the a posteriori distributions

A feature that all a posteriori probability distribution both on synthetic data and on real data had in common was the depth dependence of the resolution. It is clear that the response from deep layers have less impact on the data than the response from shallow layers. The reason for this is that signals which have propagated through the deep layers have traveled a longer distance, thus strength of the signal has attenuated more. These signals are therefore more drowned in the background noise. This also explains why low background noise give much better resolution of the resistivity profile (figure 4.6).

Also, if a highly resistive area is found, the inversion solution usually shows a thicker highly resistive reservoir but with lower resistivity than what was the actual case for the reference model. This is especially the case when smoothness regularization is imposed on the models.We saw, during the first tests, that we could not resolve either resistivity or thickness of the highly resistive target, but that the product of the two was well defined. This explains why the samples of the a posteriori distribution are more likely to contain broader highly resistive reservoirs with lower resistivity magnitude, especially when smoothness regularization is included. The results showing that the resistivity-thickness product of a high resistive layer is better resolved in marine CSEM 1D inversion than merely the

resistivity, is confirmed in reference [24].

6.2 The resistivity distribution in models accepted as samples of the a posteriori probability distribution

The results from the first tests lead to the idea of calculating the average resistivities and the second order moment for each model sampled from the a posteriori probability distribution.

When moving on to the Monte Carlo simulations where synthetic data estimated from multi layered models were used, we saw that the models sampled from the a posteriori probability distribution rarely had the same average resistivity as the reference model (see figures A.1 in appendix A).

For instance, in the synthetic case study where there was only one highly resistive layer (section 4.2.2) at 1000.0 m below the surface, the average resistivitiy in all models sampled from the a posteriori probability distribution was too high. However, the average resistivity of the samples came significantly closer to the average resistivity of the reference value when jointly using both source frequencies 0.25Hz and 0.75 Hz.

A quotation from reference [24] reads:

> As a thumb of rule, if an anomalous body is twice as far away, it has to be twice as big to produce the same measured anomaly.

Taking a closer look at the a posteriori probability distributions in figure 4.3 we see that the total resistivity from the sea floor and down to the reservoir location is lower than the total resistivity in this depth range for the reference model. To compensate, the resistivity in the deeper layers rises above the reference value. Because the effect on the data due to the resistivities in deep layers is less than for shallower layers, the additional resistivity in the deeper layers exceed the "missing" resistivity in the shallow layers. This also explains why the average resistivity over all layers is concentrated much deeper than for the reference model (see figures A.1 c and d).

For the third synthetic reference model we also saw that the average resistivity of the samples from the a posteriori distribution was too high and that the resistivity distribution was too deep (see figure A.4). However, when the smoothness regularization was included and thereby excluding a large number of non physical models, the samples had a much more correct average resistivity and second order moment.Thus, regularization can work in favor of resolving the total resistivity.

6.2.1 The effect of using different frequencies

When investigating the effect of using multiple frequencies, we found that the improvement is drastic when using two frequencies with suitable spacing instead of one. However, we also found that the resolution of the resistivity profile does not continue to improve significantly when increasing the frequency density or outer frequency limits. Because increasing the number of frequencies also increases the computation time, we prefer using few frequencies.

The effect of jointly using multiple frequencies for 1D marine CSEM inversion was also discussed in reference [22]. The conclusion in this paper is consistent with the results from section 4.2.3. Here, ten different 1D inversion results for each source frequency combination was used for the investigation instead of looking at the a posteriori distributions of models. With our method we see resolution properties of 200 000 models for each source frequency combination. This implies that the Monte Carlo method in this thesis for investigating the effect of using different frequencies provides more certain results.

From figures 4.3 and 4.5 we draw the conclusion that using high source frequencies result in better resolution of highly resistive layers and low source frequencies result in better resolution of the background resistivity in deep layers. We can also see this effect in the the plots from figure 3.3 that were obtained for the three layered first test case in section 3.1.2. Here, the a posteriori distributions when using a 0.5Hz source frequency has a tail towards toward the shallow depths whereas the a posteriori distribution when using 0.75Hz has a tail toward the deep layers. The tail indicate that the resistivity values are more poorly resolved at this depth. Consequently, using a 0.5 Hz source signal resolves the resistivities in deep layer better than a 0.75 Hz source frequency, and vice versa.

The anomalous response from a reservoir will only be detected by the receivers at the sea floor if the perturbations are not masked by the background currents [25]. Low frequencies result in little attenuation (see equation 2.1) and therefore the background currents are strong. Consequently, the resolution of a potential reservoir becomes poor. For higher frequencies the current rapidly attenuates, and the background current becomes small at much shorter offsets. Thus, it makes sense that a highly resistive layer will be better resolved when using higher source frequencies. However, if the the source frequency is too high, the currents are severely attenuated before reaching the highly resistive target. Thus, the interaction between the source signal and the reservoir is not significant, and the receivers will not detect the reservoir even though the background currents are negligible at the sea floor.

The optimal source frequency depends on the resistivity structure in the seabed which is the reason why there is not one standard frequency used

for marine CSEM surveys [25].

Low source frequencies resolves the background resistivity better according to reference [26], but 0.25 Hz is not low enough for this purpose. In reference [26] it is argued that frequencies as low as 0.05 Hz or less resolve the deep background resistivity. Only medium and high frequencies (0.25 Hz, 0.75 Hz 1.25 Hz) were used in the data acquisition on the Troll oil field and this could partly explain the unlikely high background resistivity values.

6.3 The use of a priori information

In this thesis we have also seen examples of how the a priori information imposed on the solutions influence the a posteriori probability distributions. In section 4.4.2 the smoothness requirement resulted in a much better recovery of the deepest highly resistive layer. When no smoothness regularization was used, the deepest target was not resolved at all. In this case the smoothness constraint served its purpose of excluding non physical models.

The shallow highly resistive target was better resolved when the only a priori information assumed was the sampling described in section 3.2.3. A depth dependent regularization, however, did not significantly improve the resolution for this highly resistive target compared to the results obtained with depth independent regularization (see figures A.5 and 4.8b respectively) This indicates that constraining the resistivity in deep layers can lead to less resolved shallow layers.

6.4 Troll West Oil Province

The first thing to comment on our results from the Troll oil field is the highly resistive area found at around 432-832 m below the surface in both the start model and the a posteriori distributions. Other marine CSEM inversion results also show this highly resistive top area. An inversion result obtained from common midpoint (CMP) inversion is shown in figure A.6. Also, in reference [27] high resistivity at this depth was found to be consistent with measurements from well logs. Here it is stated that this is the result of a so-called glacio-marine sediment package.

Assuming isotropic models, the a posteriori probability distribution did not vary much with and without a smoothness regularization. The main difference with and without the smoothness constraint was seen at 2000 - 3000 m below the surface. This means that the resolution power of the data was weak enough at this depth for the regularization term to significantly influence the result. In contrast, the resistivity in the top layers was not in-

fluenced by the regularization. Here, the resolution power of the data have been strong enough for the effect of the priori information to be minimal.

When allowing anisotropic models and including no smoothness or anisotropy constraints the horizontal resistivities were in general poorly resolved (see figure 5.8b). The only well defined horizontal resistivity values were low resistivity values. Thus, the data must be much more sensitive to horizontal high conductivities than horizontal high resistivities, which is consistent with the theory in section 2.1. Therefore, the horizontal components helps resolving the resistivity in the layers just above and below a highly resistive target.

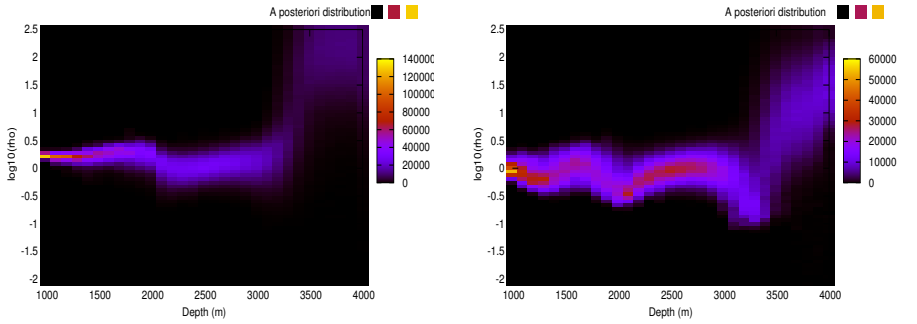
The a priori information had a huge impact on the a posteriori distribution of models when allowing anisotropic models. When excluding most models with higher horizontal than vertical resistivity values and imposing smoothness, the a posteriori probability distribution of horizontal resistivity values became significantly sharper. The horizontal a posteriori distribution also followed the same path as the vertical resistivity distribution.

Constraining the models further did not have a huge impact on the results, but it was noticeable that the horizontal and vertical resistivities became even more alike, especially in deep layers (> 2000m). Although the smoothness regularization was equal for both the isotropic and anisotropic simulations, the anisotropic case (when constraining some anisotropic properties) gave a much more defined, smoothened a posteriori distribution of models than for the isotropic case (compare figures 5.9 and 5.10 with figure 5.7). We therefore conclude that anisotropic inversion is better than isotropic inversion when constraining anisotropic properties.

Because 1D models assume infinite x and y directions no 1D model will truly explain the data from a 3D reference model. Trying to fit data acquired from a 3D reference model to 1D models will especially give less reliable results for small reference reservoirs. The Troll oil field is small enough for 1D models to not be able to fully describe the data, but large enough for the oil reservoir to be identified at depth although a few hundred meters too shallow.

To check that 1D inversion on data acquired from a 3D reference model can place a highly resistive target slightly too shallow, the Monte Carlo simulation was run on data acquired from a synthetic 3D reference model. This reference model consisted of a 100 m thick body with vertical resistivity $\rho_v = 25\Omega m$ at 1000.0 m below the sea floor. The lateral extent of the highly resistive body was 3×4 km, the background vertical resistivity was $\rho_v = 2.0\Omega m$ and the anisotropy ratio was $\lambda^2 = 2.0$. The water depth of the 3D reference model was 1000.0 m and the water resistivity was $\rho = 0.32\Omega m$. In the Monte Carlo simulation smoothness regularization and a high misfit penalty on $\rho_v < \rho_h$ was included. The resulting a posteriori distribution of models can be found in figure 6.1. For the vertical resistivity we see a resistivity contrast at 2100.0 m below the surface which

Figure 6.1 A posteriori information on synthetic 3D reference model



a) Vertical resistivity b) Horizontal resistivity

is exactly where the lower boundary of the highly resistivity body in the reference model is placed. Above this depth, we see a slightly higher resistivity bump at 1500.0 m to 2100.0 m. For the horizontal resistivity the effect of the highly resistive body is better seen, but it is clearly placed a few hundred meters too shallow. The resistivity-bump at around 1500.0 to 2100.0 m is much lower than the resistivity value in the reference value. This effect is much more significant here than when using synthetic 1D reference models. Thus, we should also expect that the highly resistive areas found by using the TWOP data, actually has much greater resistivity. We also notice that the resistivity (both vertical and horizontal) in deep layers (>3200.0 m below the surface) became very high, indicating that some high resistivities at shallower depths have not been properly resolved. This indicates that the very high background resistivities found on the TWOP could also be a result of trying to fit 1D models to data acquired from a 3D model.

A question such as

> What is the probability that there exists highly resistive areas between 1300m and 1600 m below the surface at the TWOP?

can only be answered according to the a posteriori distribution of 1D models in this thesis. However, if 2D models were used, the depth of the anomalous body would be better determined and the question could be more confidently answered. Also, if the 3D reference model is very large in the x- and y-directions, a more confident answer could be given. The answer to such a question will also depend on the a priori information one chooses to use.

In tables A.1 and A.2, we see the probabilities of there being an average resistivity higher than $5\Omega m$ at a given depth interval of 300.0 m for the three

different a priori informations used on the Troll oil field in this thesis. For all vertical resistivities the most probable depth-interval of where to find a highly resistive area is at 1132-1432 m below the surface. We see that this is also the case for the horizontal resistivities when the strict misfit penalty on models where $\rho_v < \rho_h$ is included. In addition we notice that the fraction of models with high vertical resistivity at 1132-1432 m below the surface increases dramatically when also including a misfit penalty when $\lambda^2 > 3.0$.

Combining the numbers in tables A.1 and A.2 with our results that 1D inversion using a 3D reference model reproduces a highly resistive reservoir a few hundred meters too shallow and with lower resistivity, we suggest that the results from tables A.1 and A.2 are more realistic if we move them a 100.0-200.0 m meters deeper. Then the conclusion becomes that the existence of a highly resistive area at around 1500.0 m below the surface is very probable, especially when assuming the third a priori information described in section 5.4.

Chapter 7

Conclusions

We have seen that a Monte Carlo method for sampling solutions of inverse problems can be used to display the marginal a posteriori probability distributions of sub sea resistivity models. Unfortunately it is impossible to display the joint probability distribution because of the high dimensionality of the problem.

The synthetic studies in this thesis have shown many of the same resolution properties of marine CSEM data as previously obtained from other methods of investigation. Amongst these results are the depth dependence of the resolution power of the data, the advantages of jointly using multiple frequencies and that the resistivity-thickness product of a highly resistive layer is well resolved for 1D inversion.In addition we have seen that using high frequencies for the data acquisition ($1.0 \sim 3.0Hz$) in general gives data that resolves shallow, highly resistive layers better compared to using low frequencies ($0.25 \sim 1.0Hz$). Through the sampling of models from the a posteriori distribution we were also able to confirm that the inline data are less sensitive to horizontal high resistivities than vertical high resistivities, but more sensitive to high horizontal conductivities.

In addition to conducting synthetic studies, we have also run the Monte Carlo simulation using data acquired from the TWOP. By comparing the a posteriori distribution of models using data from a synthetic 3D reference model, we have identified some limitations of explaining real data with 1D models. In our case, we found that the sampling of 1D models resulted in a reservoir placed a few hundred meters too shallow and the resistivity value of the target layer was severely weakened. Taking this information into account, we estimated the probability of the existence of a highly resistive area at depth at the TWOP given the a priori information.

Due to the increased number of parameters when moving from 1D to 2.5D and 3D models, a Monte Carlo approach for these cases will not provide much information in the near future. The method may be used to sample the a posteriori distribution just in the vicinity of an inversion re-

sult, but the information we get will be limited to very small portions of the model space. Thus, for future work, we should keep the focus on sampling 1D models. By identifying the errors made by the 1D approximation through comparison with synthetic 3D models, we can reconstruct more realistic results. Thus, this 1D model sampling is of great interest even when the data can not be fully explained by 1D models. For future work we could investigate more limitations due to using 1D models on data acquired from 3D models by conducting studies using synthetic 3D reference models of different shapes.

Because the resistivity values in deep layers were worse resolved than in the top layers one could consider to increase the thickness of each layer with depth. It would also be interesting to do more tests regarding anisotropy, both on synthetic reference models and on real data. In addition, we could jointly use marine CSEM- and magnetotelluric (MT) data in the Monte Carlo simulations, as this will provide more data directly affected by the subsea resistivity profile. This should improve the resolution of a potential highly resistive target significantly according to reference [28]. Although seismic methods provides information about other subsea properties than the resistivity, we can also use the results from seismic interpretations directly in CSEM inversion. One suggestion is to use the horizons separating different rock types and sediments, seen from seismic interpretations, as boundaries for the layers, or impose the horizons as a priori information through a soft constraint.

Bibliography

[1] K.V Sickle and J.E.Valusek. AVO analysis of 3-D seismic data identifies untested reservoirs in old gas field. *The leading edge*, 9:18–22, 1990.

[2] D. Wright, A.Ziolkowski, and B.Hobbs. Hydrocarbon detection and monitoring with a multicomponent transient electromagnetic (MTEM) survey. *The leading edge*, 21:852–864, 2002.

[3] AA.P Thirud. Waves of information. EMGS article,Scandinavian Oil-Gas Magazine, No.3/4, pp.8-9, 2002.

[4] S. Ellingsrud, T.Eidesmo, and S.Johansen. Remote sensing of hydrocarbon layers by seabed logging (SBL): Results from a cruise offshore Angola. *The leading edge*, 21:972–982, 2002.

[5] Constable S. and L.J.Srnka. An introduction to marine controlled-source methods for hydrocarbon exploration. *Geophysics*, 72:WA3–WA12, 2007.

[6] A. Tarantola. Popper, Bayes and the inverse problem. *Nature Physics*, 2:492–494, 2006.

[7] J. Chen, G.M Hoversten, D.Vasco, Y. Rubin, and Z.Hou. A Bayesian model for gas saturation estimation using marine seismic AVA and CSEM data. *Geophysics*, 72:WA85–WA95, 2007.

[8] T.Eidesmo, S.Ellingsrud, L.M MacGregor, S.Constable, M.C Sinha, S.Johansen, F.N Kong, and H.Westerdahl. Sea Bed Logging (SBL), a new method for remote and direct identification of hydrocarbon filled layers in deepwater areas. *First Break*, 20:144–152, 2002.

[9] A. Hordt, P.Andrieux, F.M.Neubauer, H.Ruter, and K.Vozoff. A first attempt at monitoring underground gas storage by means of time-lapse multichannel transient electromagnetics. *Geophysical Prospecting*, 48:489–509, 2000.

[10] S. Constable and C.J.Weiss. Mapping thin resistors and hydrocarbons with marine EM methods: Insights from 1D modeling. *Geophysics*, 71:G43–G51, 2006.

[11] S.H Ward and G.W.Hohmann. Electromagnetic theory for geophysical applications. *Electromagnetic Methods Applied Geophysics*, 1:131–311, 1988.

[12] X. Lu and C. Xia. Understanding Anisotropy in Marine CSEM Data. *SEG Expanded abstract*, 26:633–637, 2007.

[13] A. Tarantola. *Inverse problem theory and methods for model parameter estimation*. Society for Industrial and Applied Mathematics, 2005.

[14] K. Mosegaard and A. Tarantola. Monte Carlo sampling of solutions to inverse problems. *Journal of Geophysical Research*, 100:12431–12447, 1995.

[15] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Sprinter-Verlag, 2004.

[16] D. Stauffer. Econophysics - A new area for computational statistical physics? *International journal of modern physics C*, 11:1081–1087, 2000.

[17] M. Hjorth-Jensen. Lecture notes on Computational physics. http://www.uio.no/studier/emner/matnat/fys/FYS3150/h08 /undervisningsmateriale/Lecture%20Notes/lecture2008.pdf.

[18] Metropolis N., A.W. Rosenbluth, M.N. Rosenbluth, A.H Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of chemical physics*, 21:1087–1092, 1953.

[19] C. Fehily. *Visual QuickStart Guide Python*. Peachpit Press, 2002.

[20] S. Kirkpatrick, C.D Gelatt Jr., and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[21] Birsan M. A byesian approach to electromagnetic sounding in a marine environment. *IEEE Transactions on Geoscience and Remote Sensing*, 41:1455–1460, 2003.

[22] K. Key. 1D inversion of multicomponent, multifrequency marine CSEM data: Methodology and synthetic studies for resolving thin resistive layers. *Geophysics*, 74:F9–F20, 2009.

[23] M.E. Everett and S. Constable. Electric dipole fields over an anisotropic seafloor:theory and application to the structure of 40 Ma Pacific Ocean lithosphere. *Geophysical Journal International*, 136:41–56, 1999.

[24] N.B. Christensen and K. Dodds. 1D inversion and resolution analysis of marine CSEM data. *Geophysics*, 72:WA27–WA38, 2007.

[25] E.S Um and D.L. Alumbaugh. On the physics of the marine controlled-source electromagnetic method. *Geophysics*, 24:WA13–WA26, 2007.

[26] P. Dell'Aversana, M. Vivier, and A.Tansini. Expanding the Frequency Spectrum in Marine CSEM Applications. EGM 2007 International Workshop, 2007.

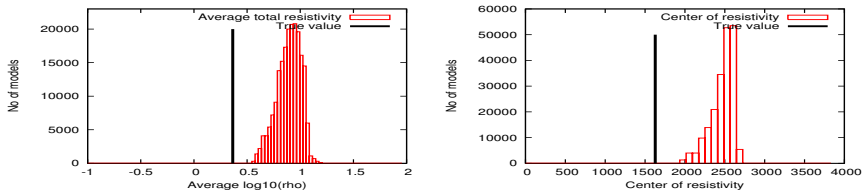
[27] I. Brevik, P.T.Gabrielsen, and J.P Morten. The role of EM rock physics and seismic data in integrated 3D CSEM data analysis. SEG Houston 2009 International Exposition and Annual Meeting, 2009.

[28] L. MacGregor and P. Harris. Marine CSEM Sounding: Moving beyond the Image. EGM 2007 International Workshop, 2007.

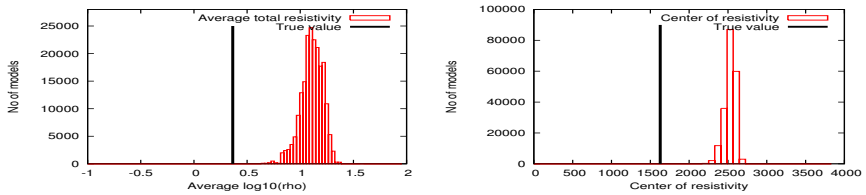
Appendix A

Additional figures

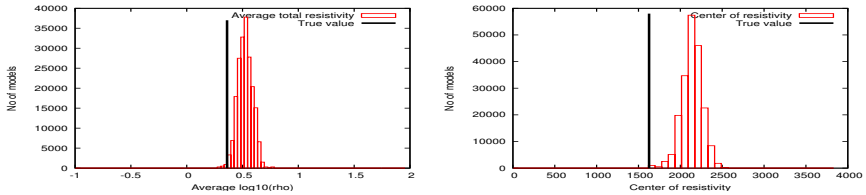
Figure A.1 Distribution of average resistivities and center of resistivities for models sampled from the a posteriori probability distribution.This figure was obtained using the first synthetic reference model in chapter 4



a) The distribution of total resistivity and center of resistivity for a source frequency of 0.25 Hz

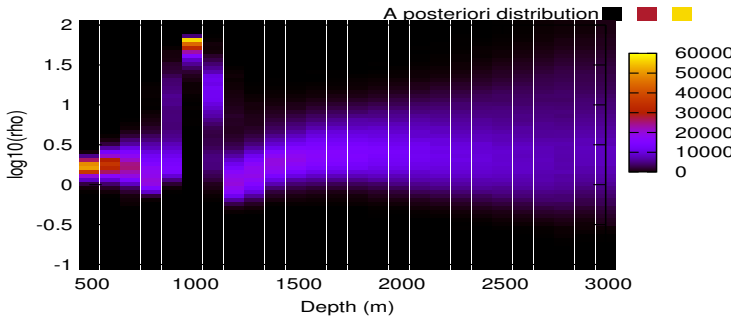


b)The distribution of total resistivity and center of resistivity for a source frequency of 0.75 Hz

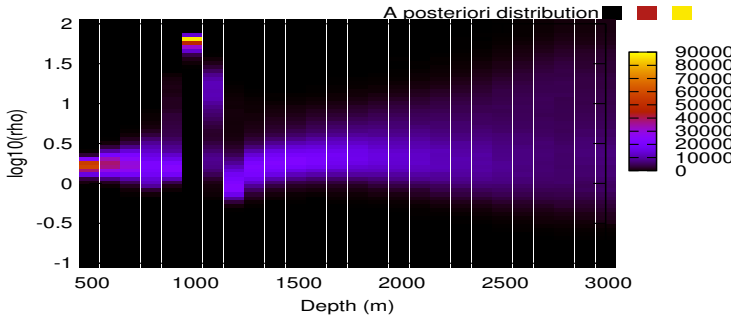


c)The distribution of total resistivity and center of resistivity for jointly using source frequencies 0.25 Hz and 0.75 Hz

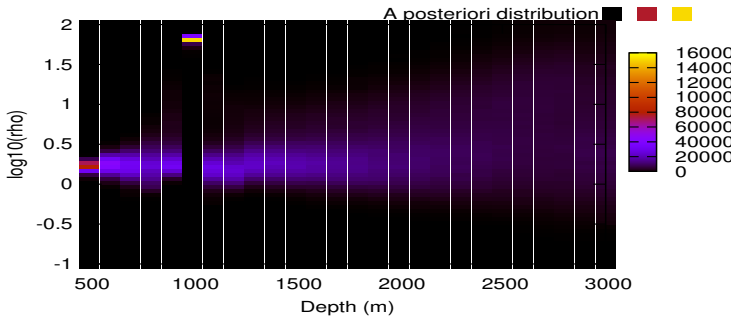
Figure A.2 Marginal a posteriori distribution



a)frequencies: 0.25 Hz, 1.0 Hz



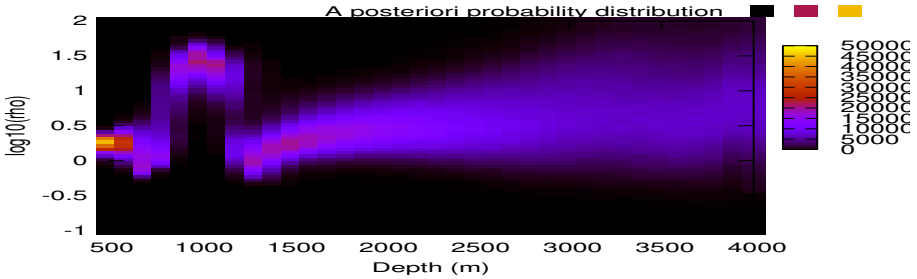
b) frequencies: 0.25 Hz, 0.50 Hz, 0.75 Hz, 1.0 Hz



c) frequencies: 0.25 Hz, 1.0 Hz, 3.0 Hz

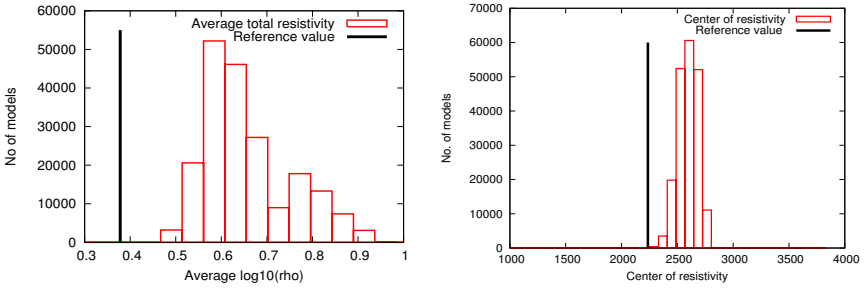
The results are obtained when using different source frequency combinations

Figure A.3 Marginal a posteriori probability distribution

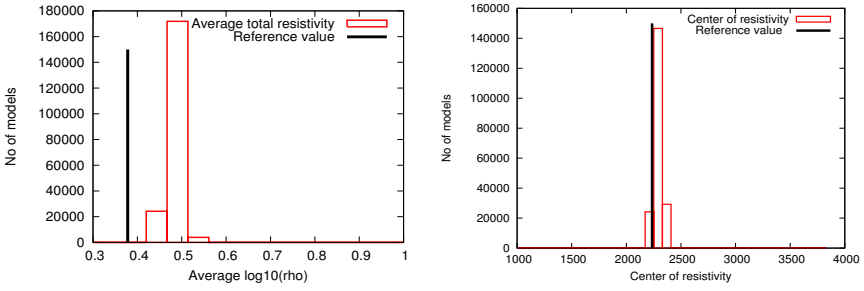


The result is obtained when starting the Monte Carlo simulation from a model consisting of a uniform subseafloor resistivity structure of $2.0\Omega m$ and jointly using both 0.25 Hz and 0.75Hz as source frequencies.

Figure A.4 Distribution of average resistivities for the models sampled from the a posteriori probability distribution. This figure was obtained using the third synthetic reference model in chapter 4



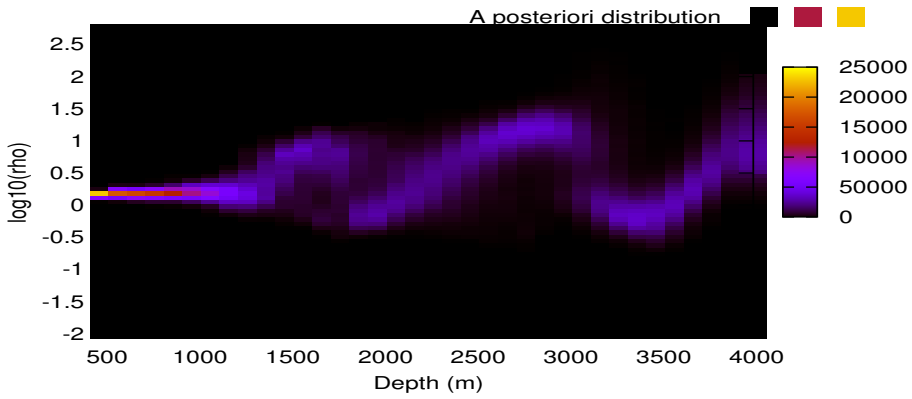
a) No regularization



b) Strict regularization

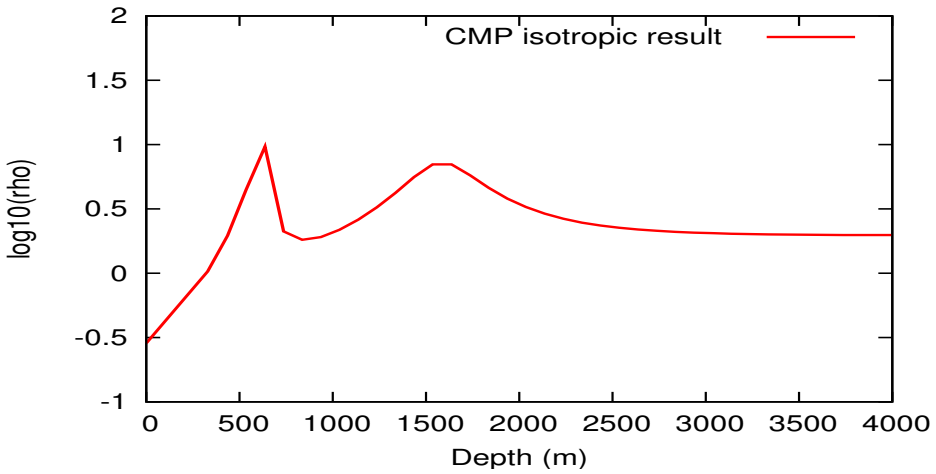
The models sampled from the a posteriori probability distribution when smoothness is not assumed have average resisitivities closer to the average resistivity for the reference model.

Figure A.5 Marginal a posteriori probability distribution



The result is obtained when using the depth dependent regularization in equation 4.9.

Figure A.6 CMP inversion result on TWOP



This resistivity profile on the TWOP was found by CMP inversion by someone else at EMGS

Table A.1: Probabilities vertical resistivities

Probability	No smoothness regularization	Smoothness and penalty on $\rho_v < \rho_h$	Smoothness and penalty on $\rho_v < \rho_h$ and $\lambda^2 > 3.0$
$P(\int_{832}^{1132} \rho dz > 1500.0)$	0.0	0.0	0.0
$P(\int_{932}^{1232} \rho dz > 1500.0)$	0.0	0.00117	0.00690
$P(\int_{1032}^{1332} \rho dz > 1500.0)$	0.03772	0.06026	0.28815
$P(\int_{1132}^{1432} \rho dz > 1500.0)$	0.10515	0.28567	0.92635
$P(\int_{1232}^{1532} \rho dz > 1500.0)$	0.02409	0.05670	0.55506
$P(\int_{1332}^{1632} \rho dz > 1500.0)$	0.00403	0.00238	0.07418
$P(\int_{1432}^{1732} \rho dz > 1500.0)$	0.00442	0.0	0.0
$P(\int_{1532}^{1832} \rho dz > 1500.0)$	0.01031	0.0	0.0

Table A.2: Probabilities horizontal resistivities

Probability	No smoothness regularization	Smoothness and penalty on $\rho_v < \rho_h$	Smoothness and penalty on $\rho_v < \rho_h$ and $\lambda^2 > 3.0$
$P(\int_{832}^{1132} \rho dz > 1500.0)$	0.995	0.0	0.0
$P(\int_{932}^{1232} \rho dz > 1500.0)$	0.99803	0.00239	0.0
$P(\int_{1032}^{1332} \rho dz > 1500.0)$	0.99661	0.04241	0.00958
$P(\int_{1132}^{1432} \rho dz > 1500.0)$	0.86994	0.11307	0.07511
$P(\int_{1232}^{1532} \rho dz > 1500.0)$	0.55732	0.01725	0.01444
$P(\int_{1332}^{1632} \rho dz > 1500.0)$	0.54063	0.00145	0.00090
$P(\int_{1432}^{1732} \rho dz > 1500.0)$	0.87744	0.0	0.0
$P(\int_{1532}^{1832} \rho dz > 1500.0)$	0.96334	0.0	0.0

Appendix B

Source codes

B.1 Script for the first tests

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

import sys
import os
import os.path

import shutil
import loghelper
import arguments
import pyelio #Needed to read files of type <filename>.nc
import random
import math

progName = "firstMC.py"
infostring = """
NAME:filtesting.py
DESCRIPTION:
The program constructs several models consisting of one resistive layer in a uniform background.
The resistive layer varies in depth, thickness and resistivity.
Synthetic data are esitmated for the different models by calling elcardinal,
and then compared with the data from the obsResult file (observed data)

The resultfile, <filename>.dat,  contains columns of data representing:
Likelihood value |  log(resistivity of the layer) | depth of upper boundaray of the layer (from surface) |

""" #end multiline string


def main(argv):
    if('-help' in argv):
        print infostring
        sys.exit(0)
    optionList = [
        'obsResults=infiltest2.nc', 'File containing observed data',
        'filename=result',          'The name of the resultfile',
        'nrmodels=1000 ',           'Number of accepted models',
        'resmaxhor=200.0',          'Upper bound for the horizontal resistivity of the layer',
        'resminhor=1.0',            'Lower bound for the horizontal resistivity of the layer',
        'resmaxver=200.0',          'Upper bound for the vertical resistivity of the layer',
        'resminver=1.0',            'Lower bound for the vertical resistivity og the layer'.
        'maxdepth=700.1',           'Max depth to the upperborder of the resistive layer',
```

```
    'mindepth=699.9',           'Min depth to the upperborder of the resistive layer',
    'thickmax=500.0',           'Max thickness of the resistive layer',
    'thickmin=5.0',             'Min thickness of the resistive layer.',
    'waterreshor=.3125',        'Resistivity of the water in the horizontal direction',
    'waterresver=.3125',        'Resistivity of the water in the vertical direction',
    'bgresver=2.0',             'Vertical resistivity of the background',
    'bgreshor=2.0',             'Horizontal resistivity of the background',
    'waterdepth=200.0',         'Depth to seafloor',
    'par=res',                  'par=con for conductivity input, par=res for resistivity input',
    'tvi=off',                  'Isotropic models if tvi=off. Can also use tvi=on',
    'skip=0',                   'Number of accepted models to skip in the beginning of the simulation',
    'alpha=0.05',               'The multiplicative data uncertainty',
    'noise=-15',                'The noise level. 10^noise' ,
    ]


arg  = arguments.arguments(optionList,argv)
vars = arg.getargs()

obsResults  = vars['obsResults']
filename    = vars['filename']
nrmodels    = int(vars['nrmodels'])
resmaxhor   = float(vars['resmaxhor'])
resminhor   = float(vars['resminhor'])
resmaxver   = float(vars['resmaxver'])
resminver   = float(vars['resminver'])
mindepth    = float(vars['mindepth'])
maxdepth    = float(vars['maxdepth'])
waterreshor = float(vars['waterreshor'])
bgreshor    = float(vars['bgreshor'])
waterresver = float(vars['waterresver'])
bgresver    = float(vars['bgresver'])
waterdepth  = float(vars['waterdepth'])
thickmin    = float(vars['thickmin'])
thickmax    = float(vars['thickmax'])
par         = vars['par']
tvi         = vars['tvi']
skip        = int(vars['skip'])
alpha       = float(vars['alpha'])
noise       = float(vars['noise'])
noise       = math.pow(10,noise)
frequencies = []

if (resmaxhor < resminhor) or (resmaxver < resminver) or (maxdepth < mindepth) or (thickmax < thickmin) :
    print 'Error: The max and min values are incorrect'
    sys.exit(0)


#COLLECTING THE FREQUENCIES FROM THE OBSERVED DATA FILE
obsr   = pyelio.SurveyResult(obsResults, pyelio.FileOpenMode.READ_ONLY)
ex1obs = obsr.getChannel(pyelio.ChannelType.CH_EMF_EX,1)
if ex1obs.hasFrequencyResult() :
    freqsobs = ex1obs.getFrequencyResult().getFrequencies()
    for f in freqsobs :
        frequencies.append(f.getValue())
        print f.getValue()

#OPEN RESULTFILE
raccept = open(filename, 'w')

#ALLOCATIONS
resaccept   = []
```

```
depthaccept = []
thickaccept = []
probaccept  = []

model           = 0
probability_old = 0.0
reject          = 0
rejecttot       = 0
accept          = 0
accepttot       = 0
steplength      = 0.1
nrmoves         = 0
case            = 0

tvi_off_on = 3
if (tvi=='on'):
    tvi_off_on = 4

#CREATING STARTMODEL
rdepth  = random.random()
rreshor = random.random()
rthick  = random.random()
rresver = random.random()

resistivityhor_old = resminhor + (resmaxhor-resminhor)*0.5 #rreshor
resistivityver_old = resminver + (resmaxver-resminver)*0.5 #rresver
depth_old          = mindepth  + (maxdepth -mindepth)*0.5 #rdepth
thick_old          = thickmin  + (thickmax-thickmin)*0.5 #rthick
layerlower_old     = depth_old + thick_old

zlay_old = [waterdepth, depth_old, layerlower_old]
filename = 'model' + str(model) + '.nc'
zr       = zlay_old[0]
mhor     = [waterreshor,bgreshor,resistivityhor_old, bgreshor]
mver     = [waterresver,bgresver,resistivityver_old, bgresver]
#END CREATION OF STARTMODEL

cmd=mkcommand(filename,par, zlay_old, frequencies, tvi, mhor, mver, zr)
loghelper.runCommand(cmd)
misfit_old      = calc_misfit_ALL_FREQS(model,freqsobs,alpha,noise)
probability_old = math.exp(-misfit_old)

#BEGINNING MONTE CARLO LOOP
while(accepttot < nrmodels + skip) :
    print 'accepttot', accepttot
    nrmoves+=1
    resinrangehor = False
    resinrangever = False
    depthinrange = False
    thickinrange = False

    #TAKING A STEP
    if (case==0):

        resistivityver=new_model(resinrangever,resmaxver,resminver,resistivityver_old,steplength)
        resistivityhor=resistivityhor_old
        depth=depth_old
        thick=thick_old

    if(case==1):

        depth= new_model(depthinrange, maxdepth, mindepth, depth_old, steplength)
```

```
    resistivityhor=resistivityhor_old
    resistivityver=resistivityver_old
    thick=thick_old

if(case==2):

    thick= new_model(thickinrange, thickmax, thickmin, thick_old, steplength)
    resistivityhor= resistivityhor_old
    resistivityver=resistivityver_old
    depth=depth_old

if (case==3):

    resistivityhor=new_model(resinrangehor,resmaxhor,resminhor,resistivityhor_old,steplength)
    resistivityver=resistivityver_old
    depth=depth_old
    thick=thick_old
#END STEP

#THE VALUES TO BE SENT TO ELCARDINAL
zlay     = [waterdepth, depth, depth + thick]
mhor     = [waterreshor,bgreshor,resistivityhor, bgreshor]
mver     = [waterresver,bgresver,resistivityver, bgresver]
filename = 'model' + str(model) + '.nc'

#RUNNING ELCARDINAL
cmd  = mkcommand(filename,par, zlay, frequencies, tvi, mhor, mver, zr)
loghelper.runCommand(cmd)
misf = calc_misfit_ALL_FREQS(model, freqsobs,alpha,noise)

# FIND PROBABILITY OF MODEL
probability = math.exp(-misf)

#UPDATE STEPLENGTH
if (nrmoves%20==0):
    print 'accept=', accept, 'reject=',  reject, "nrmoves=", nrmoves
    steplength=adjust_steplength(accept, reject, steplength)
    accept=0
    reject=0

delta = (-misf + misfit_old)
r     = random.random()

if ((-misf + misfit_old) > 500.0):
    delta = 10.0
    r     = 1.0

#METROPOLISTEST
if( misf < misfit_old or r < math.exp(delta) ):
    print 'ACCEPT'
    accepttot +=1
    accept    += 1
    probability_old    = probability
    resistivityhor_old = resistivityhor
    resistivityver_old = resistivityver
    thick_old          = thick
    depth_old          = depth
    misfit_old         = misf

    if (accepttot > skip):
        resaccept.append(resistivityver_old)
        depthaccept.append(depth_old)
```

```
                thickaccept.append(thick_old)
                probaccept.append(probability_old)

        else :
            reject +=1
            rejecttot+=1
            print 'REJECT'
        #END METROPOLIS TEST
        #REMOVE FILES
        removefile = 'rm ' + filename
        loghelper.runCommand(removefile)

        if(accepttot==skip):
            rejecttot = 0
            model     = 0

        #SET CASE: NEW PARAMTER TO VARY
        case+=1
        case = case % tvi_off_on
        model += 1

    print 'rejecttot =', rejecttot , ' accepttot= ', accepttot
    #PRINT TO RESULTFILE
    for k in range(len(resaccept)):
        raccept.write(str(probaccept[k]) + '\t' + str(resaccept[k]) + '\t'
                      +  str(depthaccept[k]) + '\t'+ str(thickaccept[k]) + '\n')

    raccept.close()
##########################################################################################################################

def mkcommand(filename,par, layers, frequencies, tvi, mhor,mver, zr):
    if (tvi=='off'):
        mhor=mver

    if (len(layers)!= len(mver)-1) :
        print 'Error: Each layer needs a spesified value for the resistivity/conductivity'
        print len(layers), len(mver)

    zs= zr - 30

    command = 'elcardinal '
    command += 'ofel=' + filename
    command += ' par=' + par
    command += ' zlay='
    for i in range(len(layers)-1):
        command+= str(layers[i]) + ','
    command += str(layers[len(layers)-1])
    command += ' freq='
    for i in range(len(frequencies) -1):
        command+= str(frequencies[i]) + ','
    command +=  str(frequencies[len(frequencies)-1])
    command += ' tvi=' + tvi
    command += ' mver='
    for i in range(len(mver)-1):
        command +=  str(mver[i]) + ','
    command += str(mver[len(mver) - 1])
    command += ' mhor='
    for i in range(len(mhor)-1):
        command +=  str(mhor[i]) + ','
    command += str(mhor[len(mhor) - 1])
    command += ' zr='+ str(zr)
    command += ' zs='+ str(zs)
```

```
    command += ' comp=' + 'ex'

    return command

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

def adjust_steplength(accept, reject, steplength):

  if (float(accept)/float(reject+accept) < 0.50):
      steplength/=1.1
      print 'step length decreased. accept < 50%', 'stepl= ', steplength

  else:
      steplength*=1.1
      print 'step length increased. accept >= 50%', 'stepl= ', steplength
  return steplength

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

def new_model(bool, max, min,pos_old, steplength):
     while not bool:
            r=random.random()
            step = (max-min)*steplength
            pos_new = pos_old + (r-0.5)*(step)
            bool=True
            if(pos_new < (min) or pos_new > (max)) :
                bool=False
     return pos_new

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


def calc_misfit_ALL_FREQS(model,freqsobs,alpha,noise):
    sr = pyelio.SurveyResult('model' + str(model) + '.nc', pyelio.FileOpenMode.READ_ONLY)

    ex1=sr.getChannel(pyelio.ChannelType.CH_EMF_EX,1)
    if ex1.hasFrequencyResult():
        ex1_freq = ex1.getFrequencyResult()
        freqs = ex1_freq.getFrequencies()
    misf= 0

    #Calculate misfit: |Ex(offset)-Ex_obs(offset)|^2/(|Ex|^2 + eta)
    for i in range(len(freqsobs)):
        data = list(freqs[i].getComplexSamples())
        dataobs = list(freqsobs[i].getComplexSamples())

        for k in range(len(dataobs)):
            misf += (abs(data[k]-dataobs[k])*abs(data[k]-dataobs[k]))
    misf /=(alpha*alpha*(abs(dataobs[k])*abs(dataobs[k]))+noise*noise)
    return misf

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

#############################################################################################################
if __name__=="__main__":
    main(sys.argv)
#############################################################################################################
```

B.2 Script for parallel computing

```
#! /usr/bin/env python
import sys
import os.path
#Set dir for local modules.
binDir = os.path.dirname(sys.argv[0])
binDir = os.path.abspath(binDir)+'/'
sys.path.append(binDir) # Where to find the following modules:

import arguments
import elgogo

progName = "parallelMC.py"
infostring = """
NAME: parallelMC.py
The script calls the multilayered Monte Carlo code multiMC.py in parallel.
The number given for paraJobs defines how many jobs are to be run in parallel.
When all jobs are done, the script calls the program collect.py which
collects the results from all jobs and put the resulting files in the directory
resultDir/collectedFiles.Default directory for the resulting files is
Troll/collectedFiles.The file containing the startmodel must consist of
three columns containing:

|Depth |Vertical resisitivity | Horizontal resisitivy|

""" #end multiline string

def main(argv):
    if('-help' in argv):
        print infostring
        sys.exit(0)

    optionList = [
        'prefix=Troll',              'Prefix for the parallel job',
        'ofile=Tr_007_Data.nc',      'The file containing observed data',
        'resultDir=Troll',           'Where to store the result files.',
        'waterdepth=332.00',         'Depth to seafloor',
        'maxdepth=4332.0',           'Depth of the deepest layer boundary',
        'nrzlay=41',                 'The number of layers below the seafloor',
        'maxcon=100.0',              'The maximum conductivity',
        'mincon=0.002',              'The minimum conductivity',
        'mverstart=3.2,0.5',         'Vertical water conductivity:water,background',
        'mhorstart=3.2,0.5',         'Horizontal water conductivity',
        'nrvar=1000',                'The number of whole iterations',
        'tvi=on',                    'tvi=on gives anisotropic models and tvi=off isotropic ',
        'skip=1000',                 'The number whole iterations to skip in the beginning',
        'alpha=0.025',               'The multiplicative data uncertainty',
        'noisefile=Tr_007_Noise.nc', 'Input noise file.If noisefile=none 10^noisefloor value is used ',
        'noisefloor=-15',            'Noise level. 10^noisefloor'  ,
        'acur=4',                    'Accuracy in integration for the forward problem',
        'startmod=startmodel',       'File with startmodel.If "none",the halfspace model is used',
        'paraJobs=100',              'Number of jobs to run in parallel',
        'randomSeedGroup=1',         'randomSeed=randomSeedGroup*654321+i*1234+7',
        ]


    # read command line
    arg = arguments.arguments(optionList,argv)
    vars = arg.getargs()

    prefix          = vars['prefix']
```

```
resultDir       = vars['resultDir']
kFaktor         = vars['kFaktor']
randomSeedGroup = vars['randomSeedGroup']

# create resultDir if not exist
if (not os.path.isdir(resultDir)):
    print("# frechall_Parallel.py: Creating new directory: " + resultDir)
    os.mkdir(resultDir)
if (not os.path.isdir(resultDir+ '/collectedFiles')):
    print("# frechall_Parallel.py: Creating new directory: "
          + resultDir + '/collectedFiles')
    os.mkdir(resultDir+'/collectedFiles')


# create job list
verbose = "1"
jobList = []
jobListCollect = []
for i in range(int(kFaktor)):
    command  = binDir + "multiMC.py"
    command += " obsfile="+vars['obsfile']
    command += " noisefile="+vars['noisefile']
    command += " waterdepth="+vars['waterdepth']
    command += " maxcon="+vars['maxcon']
    command += " mincon="+vars['mincon']
    command += " nrzlay="+vars['nrzlay']
    command += " maxdepth="+vars['maxdepth']
    command += " acur="+vars['acur']
    command += " mverstart="+vars['mverstart']
    command += " mhorstart="+vars['mhorstart']
    command += " startmod="+vars['startmod']
    command += " nrvar="+vars['nrvar']
    command += " tvi="+vars['tvi']
    command += " skip="+vars['skip']
    command += " alpha="+vars['alpha']
    command += " noisefloor="+vars['noisefloor']
    command += " noisefile="+vars['noisefile']
    command += " resultfilename="+resultDir+"/result."+str(i)
    command += " randomSeed="+str(int(randomSeedGroup)*654321 + i*1234 + 7)
    command += " verbose="+verbose

    jobList.append([command,"normal"])

    if (verbose == "1"):
        verbose = "0"

print jobList

#Create joblist for collecting the results

commandCollect= binDir + 'collect.py'
commandCollect+=' files='    + resultDir+"/noise."
commandCollect+=' paraJobs=' + str(kFaktor)
commandCollect+=' result='   + resultDir + '/collectedFiles/noise'
commandCollect+= " maxcon="  +vars['maxcon']
commandCollect+= " mincon="  +vars['mincon']
commandCollect+= " tvi="     +vars['tvi']
print commandCollect

jobListCollect.append([commandCollect,"normal"])
# submitt job to queue
elgogo.lsf_go(prefix,500,jobList,".")
```

```
    #Collect results in resultDir/collectFiles
    elgogo.lsf_go(prefix,100,jobListCollect,".")

    # finalize
    print "# frechall_Parallel.py: exit at end script"

#####################################################################
if __name__=="__main__":
    main(sys.argv)
#####################################################################
```

B.3 Multilayered Monte Carlo simulation

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

import sys
import os
import os.path

#Set dir for local modules.
binDir = os.path.dirname(sys.argv[0])
binDir = os.path.abspath(binDir)+'/'
sys.path.append(binDir) # Where to find the following modules:

import shutil
import loghelper
import arguments

sys.path.append( os.environ['HOME'] + os.path.sep + 'pyelio' )
import pyelio #Needed to read files of type <filename>.nc

import random
import math
import struct

import pynavutil
#import decimal


progName = "multiMC.py"
infostring = """
NAME:multiMC.py

The script performs a Monte Carlo simulation for 1D models. Horizontal layers with fixed spacing is
assumed.The script reads data from a file of the type <filename.nc> which is the file containing
the obtained data from a true model.It then generates a series of sub seafloor models and uses
elcardinal to calulate new data, which again gives a misfit with the data from the true model.

The misfit is used in a metropolis rule, and the result is samples of the a posterior probability
distribution.
The output files are:

*****************************************************************************************************
# <resultfilename>.dat:                                                                             #
# This file contains columns with values for :                                                      #
#                     **** Vertical resistivity ****                                                #
# |depth | best model |avrage model |mean model |average log10rho*log10rho | skewness | kurtosis|      #
```

```
# If anisotrop models, then the file also contain these quantities for the horizontal resistivities    #
*****************************************************************************************************

*****************************************************************************************************
# <resultfilename>res.dat:                                                                            #
# This file contains columns with values for :                                                        #
# |depth |Average total resistivities | Second order moment of the resistivity |                     #
*****************************************************************************************************


*****************************************************************************************************
# <resultfilename>cum.dat:                                                                            #
# This file contains columns with values for :                                                        #
# |resistivity | No of models 1st layer with resistivity in column 1 | No of models 2nd layer|..|last layer|    #
*****************************************************************************************************

*****************************************************************************************************
# <resultfilename>cumhor.dat:                                                                         #
#                                                                                                     #
#  Same as for <resultfilename>cum.dat but for horizontal resistivities                               #
*****************************************************************************************************

*****************************************************************************************************
# <resultfilename>lst.dat:                                                                            #
# This file contains columns with values for :                                                        #
# |resistivity values sample no. 1 [rho1, rho2 ... rho n]|                                           #
# |resistivity values sample no. 2 [rho1, rho2 ... rho n]|                                           #
etc.                                                                                                  #
*****************************************************************************************************

*****************************************************************************************************
# <resultfilename>lsthor.dat:                                                                         #
#  Same as for <resultfilename>lst.dat but for horizontal resistivities                               #
*****************************************************************************************************


""" #end multiline string
def main(argv):
    if('-help' in argv):
        print infostring
        sys.exit(0)
    optionList = [
        'ofile=Tr_007_Data.nc',     'The file containing observed data',
        'resfile=syntuni',          'Where to store the result files.',
        'waterdepth=500.00',        'Depth to seafloor',
        'maxdepth=4000.0',          'Depth of the deepest layer boundary',
        'nrzlay=10',                'The number of layers below the seafloor',
        'maxcon=100.0',             'The maximum conductivity',
        'mincon=0.002',             'The minimum conductivity',
        'mverstart=3.69,0.25',      'Vertical water conductivity:water,background',
        'mhorstart=3.69,0.5',       'Horizontal water conductivity',
        'nrvar=10',                 'The number of whole iterations',
        'tvi=on',                   'tvi=on gives anisotropic models and tvi=off isotropic ',
        'skip=0',                   'The number whole iterations to skip in the beginning',
        'alpha=0.025',              'The multiplicative data uncertainty',
        'noisefile=Tr_007_Noise.nc','Input noise file.If noisefile=none 10^noisefloor value is used ',
        'noisefloor=-15',           'The noise level. 10^noisefloor'  ,
        'acur=4',                   'Accuracy in integration for the forward problem',
        'startmod=startmodel',      'File with startmodel.If "none",the halfspace model is used',
        'randomSeed=9',             'seed for the random number generator',
        'verbose=0',                '=0,1,2 verbose level',
```

```
    ]


#READING ARGUMENTS
arg = arguments.arguments(optionList,argv)
vars = arg.getargs()


obsfile      = vars['ofile']
resfile     = vars['resfile']
noisefile   = vars['noisefile']
waterdepth  = float(vars['waterdepth'])
maxcon      = float(vars['maxcon'])
mincon      = float(vars['mincon'])
nrzlay      = int(vars['nrzlay'])
maxdepth    = float(vars['maxdepth'])
spacezlay   = int(vars['nrzlay'])
acur        = int(vars['acur'])
mverstart   = eval(vars['mverstart'])
mhorstart   = eval(vars['mhorstart'])
startmod    = vars['startmod']
nrvar       = int(vars['nrvar'])
tvi         = vars['tvi']
skip        = int(vars['skip'])
alpha       = float(vars['alpha'])
noisefloor  = float(vars['noisefloor'])
verbose     = int(vars['verbose'])

nrcum=40

if (noisefile == 'none'):
    noise = math.pow(10,noisefloor)
    noise_dat = [math.pow(10,noisefloor)]
    print noise

randomSeed = int(vars['randomSeed'])
par='con'
nrcum=40

##### Starting the script #####
# initiate LogClass
log = LogClass(progName,int(verbose))
log.writeln("***** Starting *****")

# print options used
log.writeln("* Used options:")
log.printOptions(vars)

# Start the random number generator
random.seed(randomSeed)

mverwater=mverstart[0]
mhorwater=mhorstart[0]

log.write(str(mverstart)+"  "+str(mverwater))

frequencies = []


###READING NOISEFILE###
if (noisefile!= 'none'):
```

```
    noise=pyelio.SurveyResult(noisefile, pyelio.FileOpenMode.READ_ONLY)
    ex1noise=noise.getChannel(pyelio.ChannelType.CH_EMF_EX,1)
    if ex1noise.hasFrequencyResult() :
        noisef = ex1noise.getFrequencyResult().getFrequencies()


###COLLECTING FREQUENCIES USED TO OBTAIN DATA FROM TRUE MODEL#########
obsr = pyelio.SurveyResult(obsfile, pyelio.FileOpenMode.READ_ONLY)
#Ex-kanalen
ex1obs=obsr.getChannel(pyelio.ChannelType.CH_EMF_EX,1)
if ex1obs.hasFrequencyResult() :
    freqsobs = ex1obs.getFrequencyResult().getFrequencies()
    for f in freqsobs :
        frequencies.append(f.getValue())
        print f.getValue()
w=[]
data_obs=[]
noise_data=[]

###READING DATA FROM TRUE MODEL AND CALCULATING WEIGTHS: 1/(alpha^2*Eo^2 + noise^2)
###IF noise to signal ratio less than 10 then the weight=0

for i in range(len(frequencies)):
    print 'i', i
    if (noisefile !='none'):
        noise_data.append(list(noisef[i].getComplexSamples()))
    data_obs.append(list(freqsobs[i].getComplexSamples()))

for i in range(len(frequencies)):
    if (noisefile == 'none'):
        noise_data.append(noise_dat)

    w.append([0.0]*len(data_obs[i]))
    for k in range(len(data_obs[i])):
        if len(noise_data[i])>1:
            noise=abs(noise_data[i][k])
        datobs=abs(data_obs[i][k])
        weight= 1.0/((alpha*alpha)*(datobs*datobs) + noise*noise)
        w[i][k]=(weight)
        ratio=abs(datobs)/(abs(noise))
        if (ratio < 10):
            w[i][k]=0



###READING INN OFFSETS
nav_x = obsr.getChannel(pyelio.ChannelType.CH_SOURCE_X, 1)
nav_y = obsr.getChannel(pyelio.ChannelType.CH_SOURCE_Y, 1)
sx = nav_x.getFrequencyResult().getFrequencies()[0].getSamples()
sy = nav_y.getFrequencyResult().getFrequencies()[0].getSamples()

# Receiver coordinates
rec_x = obsr.prop_receiver_x
rec_y = obsr.prop_receiver_y

# using navutils offset calculatro
offsets = pynavutil.calculate_offsets(sx, sy, rec_x, rec_y)

rl = offsets[len(offsets)-1]
dr = offsets[1]-offsets[0]
r0 = offsets[0]
```

```
###CREATING START MODEL (FROM FILE OR FROM INPUT PARATMETERS)
layer=[]
if (startmod=='none'):
    for l in range(nrzlay):
        ld=waterdepth + l*(maxdepth-waterdepth)/(nrzlay-1)
        layer.append(ld)
    zr=waterdepth
else:
    depth=open(startmod, 'r')
    lay=open(startmod, 'r')
    ld=lay.read()
    nrl=len(depth.readlines())
    for i in range(nrl-2):
        layer.append(eval(ld.split()[3*(i+1)]))
    zr=layer[0]

tvimod=1
if (tvi =='on'):
    tvimod=2

laycum=[]
laycumhor=[]

ad=-round(math.log10(1.0/maxcon)/math.log10(1.0/mincon)*nrcum)
ad=int(ad)

newmax=ad*math.log10(1.0/mincon)/nrcum
maxcon= math.pow(10,newmax)


for k in range(len(layer)+1):
    laycum.append([0]*(nrcum+ad))
    laycumhor.append([0]*(nrcum+ad))


#INITIAL MODEL
mver_old=startmodel(mverstart, nrzlay, startmod,1)
mhor_old=startmodel(mhorstart,nrzlay, startmod,2)


if (tvimod==1):
    for i in range(len(mver_old)):
        mhor_old[i]=mver_old[i]

accepttot=0
rejecttot=0

### SOME ALLOCATIONS
probaccept  = []
avmverres   = [0]*len(mver_old)
avmhorres   = [0]*len(mhor_old)
cum2        = [0]*len(mver_old)
cum2hor     = [0]*len(mhor_old)

rhointegrated   = []
zrhointegrated  = []
z2rhointegrated = []

mveraccept = []
mhoraccept = []

for i in range(len(mver_old)):
```

```
     mveraccept.append(mver_old[i])
     mhoraccept.append(mhor_old[i])

it=0
probmax=0.0

maxstep = [0.1]*len(layer)*tvimod
accept  = [0]*len(layer)*tvimod
reject  = [0]*len(layer)*tvimod
zul     = layer[len(layer)-1]

lstverres=[]
lsthorres=[]

for i in range(len(layer)+1):

    lstverres.append([0]*(nrvar+skip+5))
    lsthorres.append([0]*(nrvar+skip+5))

check   = 60
mv_good = [0]*len(mver_old)
mh_good = [0]*len(mhor_old)
misfmin = 1000000

cmd=mkcommand(1, mver_old, mhor_old, layer, zr, tvi,acur, frequencies, r0,dr,rl,randomSeed)
loghelper.runCommand(cmd)
misf_old=calc_misfit(1,data_obs,alpha,noise,frequencies,r0,dr,rl,
                     mver_old,mhor_old,layer,tvimod,w,randomSeed)
misf_old_reg=calc_misf_reg(frequencies,mver_old, mhor_old,layer,tvimod)


### BEGINING OF MONTE CARLO LOOP OVER MODELS ###
while (len(probaccept) < (nrvar + skip) ):
    cont=False
    delta=[]
    delta=[0]*(tvimod*len(mver_old)-tvimod)
    acceptround=0
    rejectround=0
    for i in range(len(mver_old)):
        mv_good[i]=mver_old[i]
        mh_good[i]=mhor_old[i]
    misf_good=misf_old

    ### BEGINNING LOOP OVER ALL LAYERS (PARAMETERS) ###
    for l in range(len(layer)*tvimod):
        mv=[]
        mh=[]
        for i in range(len(mver_old)):
            mv.append(mver_old[i])
            mh.append(mhor_old[i])

        print '********L******', l
        inrange=False

        ### TAKING A STEP ###
        if (l < len(layer)):
            deltacon = step(inrange, maxcon, mincon, mver_old[l+1], maxstep[l])
            mv[l+1]*=deltacon
            delta[l]=mv[l+1]-mver_old[l+1]

        else:
            deltacon = step(inrange, maxcon, mincon,
```

```
                            mhor_old[l -len(layer)+1], maxstep[l-len(layer)])
                mh[l - len(layer) +1]*=deltacon
                delta[l]=mh[l-len(layer)+1]-mhor_old[l-len(layer)+1]


            misfreg=calc_misf_reg(frequencies,mv, mh,layer,tvimod)
            r=random.random()

            ### ACCEPTANCE TEST BASED ON REGULARIZATION ###
            if (misfreg < misf_old_reg or r < math.exp(-misfreg + misf_old_reg)):
                misf = calc_misfit_Frechet(1, data_obs,alpha, noise,frequencies,
delta,0,tvi,r0,dr,mv, mh,layer,tvimod,w,randomSeed)
                r=random.random()

                ###ACCEPTANCE TEST BASED ON DATA ###
                if (misf < misf_old or r < math.exp(-misf + misf_old)):
                    acceptround+=1
                    accepttot+=1
                    accept[l]+=1
                    misf_old=misf
                    relerr=0
                    misf_old_reg=misfreg
                    print 'ACCEPT'

                    if (tvi=='on'):
                        if (l < len(layer)):
                            mver_old[l+1]=mv[l+1]
                        else:
                            mhor_old[l-len(layer)+1]=mh[l-len(layer)+1]

                    else:
                        if (l < len(layer)):
                            mver_old[l+1]=mv[l+1]
                            mhor_old[l+1]=mv[l+1]

                else:
                    rejectround+=1
                    rejecttot+=1
                    reject[l]+=1
                    print 'REJECT'
                    if (l < len(layer)):
                        delta[l]=0
                    else:
                        delta[l]=0
            else:
                rejectround+=1
                rejecttot+=1
                reject[l]+=1
                print 'REJECT REG'
                if (l < len(layer)):
                    delta[l]=0
                else:
                    delta[l]=0

            ### CHECKING THE ERROR MADE IN THE MISFIT BY USING ELCARDINAL
            ### IF THE ERROR HIGHER THAN 10% THIS NEW MODEL IS DELETED
            ### AND RETURN TO THE MODEL WE HAD BEFORE THE LOOP OVER ALL
            ### PARAMETERS AND TRY AGAIN.

            if (acceptround==check):
                acceptround=0
                cmd=mkcommand(1, mver_old, mhor_old, layer, zr, tvi,acur,
```

```
                    frequencies, r0,dr,rl,randomSeed)
        loghelper.runCommand(cmd)
        misfelc = calc_misfit(1,data_obs,alpha,noise,frequencies,r0,
                              dr,rl,mver_old,mhor_old, layer,tvimod,w,randomSeed)
        relerr2=abs((misf_old+misf_old_reg) -(misfelc+misf_old_reg))/(misfelc+misf_old_reg)
        delta=[0]*(tvimod*len(mver_old)-tvimod)
        misf_old=misfelc

        if (relerr2 > 0.1):
            for i in range(len(mver_old)):
                mver_old[i]=mv_good[i]
                mhor_old[i]=mh_good[i]

            misf_old=misf_good
            misf=misf_old

            cont=True
            break

if (cont==True):
    cont=False
    continue


it+=1

### UPDATING STEP LENGTH
if (it%10==0):
    for l in range(len(layer)*tvimod):
        acpt=accept[l]
        rjct=reject[l]
        maxstep[l]=uppdate_step(acpt, rjct,maxstep[l])
        accept[l]=0
        reject[l]=0


### CHECKING RELATIVE ERROR AGAIN: IF ERROR > 10% :
### MODEL DELETED, RETURN TO THE MODEL PRIOR TO THE LOOP
### OVER ALL PARAMETERS AND TRY AGAIN
cmd=mkcommand(1, mver_old, mhor_old, layer, zr, tvi,acur,
              frequencies, r0,dr,rl,randomSeed)
loghelper.runCommand(cmd)
misfelc = calc_misfit(1,data_obs,alpha,noise,frequencies,
                      r0,dr,rl,mver_old,mhor_old, layer,tvimod,w,randomSeed)
relerr2=abs((misf_old+misf_old_reg) -(misfelc+misf_old_reg))/(misfelc+misf_old_reg)
print '*relerr*', relerr2
if (relerr2 > 0.10):
    for i in range(len(mver_old)):
        mver_old[i]=mv_good[i]
        mhor_old[i]=mh_good[i]
        misf_old=misf_good
    continue


### ALL PARAMETERS HAVE VARIED AND PASSED THE RELATIVE ERROR TESTS: STORING THE NEW MODEL
misf_old=misfelc
if (it > skip):
    integral=(zul - layer[len(layer)-1])*(math.log10(1.0/mver_old[len(layer)]))
    zintegral=(zul*zul - layer[len(layer)-1]*layer[len(layer)-1])
       *(math.log10(1.0/mver_old[len(layer)]))
    z2integral=(math.pow(zul,3.0) - math.pow(layer[len(layer)-1],3.0))
       *(math.log10(1.0/mver_old[len(layer)]))
```

```
        probaccept.append(misf_old)
        for m in range(len(layer)-1):
            integral+= (layer[m+1]-layer[m])*(math.log10(1.0/mver_old[m+1]))
            zintegral+= (layer[m+1]*layer[m+1]-layer[m]*layer[m])
      *(math.log10(1.0/mver_old[m+1]))
            z2integral+= (math.pow(layer[m+1],3.0)-math.pow(layer[m],3.0))
                      *(math.log10(1.0/mver_old[m+1]))

        for m in range(len(layer)+1):

            a=mver_old[m]
            b=mhor_old[m]
            avmverres[m]+=math.log10(1.0/a)
            avmhorres[m]+=math.log10(1.0/b)
            cum2[m]+=math.log10(1.0/a)*math.log10(1.0/a)
            cum2hor[m]+=math.log10(1.0/b)*math.log10(1.0/b)

            lstverres[m][len(probaccept)-1]=1.0/(mver_old[m])
            lsthorres[m][len(probaccept)-1]=1.0/(mhor_old[m])

            place=int(math.log10((1.0/mver_old[m]))/(math.log10((1.0/mincon))/nrcum) +ad)
            placehor=int(math.log10((1.0/mhor_old[m]))/(math.log10((1.0/mincon))/nrcum)+ad)

            laycum[m][place]+=1
            laycumhor[m][placehor]+=1


            if (misfmin > misf_old):
                mveraccept[m]=mver_old[m]
                mhoraccept[m]=mhor_old[m]
                probmax = misf_old


        zintegral/=2.0
        z2integral/=3.0
        zintegral/=integral
        z2integral/=integral
        rhointegrated.append(integral/(zul-waterdepth))
        zrhointegrated.append(zintegral)
        z2rhointegrated.append(z2integral)


for i in range(len(avmverres)):
    avmverres[i]/=len(probaccept)
    avmhorres[i]/=len(probaccept)
    cum2[i]/=len(probaccept)
    cum2hor[i]/=len(probaccept)


cumint=math.log10((1.0/mincon))/nrcum
n=[0]*len(mver_old)
nhor=[0]*len(mver_old)
median=[0]*len(mver_old)
medianhor=[0]*len(mver_old)


stdmverres=[0]*len(mver_old)
stdmhorres=[0]*len(mver_old)
skewmverres=[0]*len(mver_old)
skewmhorres=[0]*len(mver_old)
```

```
kurtmverres=[0]*len(mver_old)
kurtmhorres=[0]*len(mver_old)

####CALCULATING STATISTICAL PROPERTIES
for l in range(len(layer)):
    for i in range(len(probaccept)):
        stdmverres[l+1]+=math.pow((math.log10(lstverres[l+1][i])-(avmverres[l+1])),2)
        stdmhorres[l+1]+=math.pow((math.log10(lsthorres[l+1][i])-(avmhorres[l+1])),2)

        skewmverres[l+1]+=math.pow((math.log10(lstverres[l+1][i])-(avmverres[l+1])),3)
        skewmhorres[l+1]+=math.pow((math.log10(lsthorres[l+1][i])-(avmhorres[l+1])),3)

        kurtmverres[l+1]+=math.pow((math.log10(lstverres[l+1][i])-avmverres[l+1]),4)
        kurtmhorres[l+1]+=math.pow((math.log10(lsthorres[l+1][i])-avmhorres[l+1]),4)


    skewmverres[l+1]/=len(probaccept)
    skewmverres[l+1]/=(math.pow((stdmverres[l+1])/len(probaccept),1.5))
    skewmverres[l+1]*=(math.sqrt(len(probaccept)*(len(probaccept)-1))/(len(probaccept)-2))

    kurtmverres[l+1]/=len(probaccept)
    kurtmverres[l+1]/=(math.pow((stdmverres[l+1])/len(probaccept),2))
    kurtmverres[l+1]-=3

    stdmverres[l+1]/=(len(probaccept)-1)
    stdmverres[l+1]=math.sqrt(stdmverres[l+1])

    if(tvi=='on'):

        stdmhorres[l+1]/=(len(probaccept)-1)
        stdmhorres[l+1]=math.sqrt(stdmhorres[l+1])

        skewmhorres[l+1]/=math.pow(stdmhorres[l+1],3)
        skewmhorres[l+1]/=len(probaccept)

        kurtmhorres[l+1]/=math.pow(stdmhorres[l+1],4)
        kurtmhorres[l+1]/=len(probaccept)
        kurtmhorres[l+1]-=3


### WRITING RESULT TO FILE
resultname       = resfile + '.dat'
resultnameres    = resfile + 'res' + '.dat'
resultnamecum    = resfile + 'cum' + '.dat'
resultnamecumhor = resfile + 'cumhor' + '.dat'
resultnamelst    = resfile + 'lst' + '.dat'
resultnamelsthor = resfile + 'lsthor' + '.dat'

res         = open(resultname, 'w')
resres      = open(resultnameres, 'w')
cum         = open(resultnamecum, 'w')
lists       = open(resultnamelst, 'w')

if (tvi=='on'):
    cumhor= open(resultnamecumhor, 'w')
    listshor=open(resultnamelsthor,'w')

### WRITING ALL ACCEPTED MODELS TO FILE
for i in range(len(layer)):
    for j in range(len(probaccept)):
        lists.write(str(lstverres[i+1][j]) + ',\t')
    lists.write('\n')
```

```
if (tvi=='on'):
    for i in range(len(layer)):
        for j in range(len(probaccept)):
            listshor.write(str(lsthorres[i+1][j]) + ',\t')
        listshor.write('\n')


### WRITING BEST MODEL, MEAN MODEL, AVERAGE OF log10rho*log10rho , SKEW AND KURTOSIS TO FILE
res.write('0.0'+ '\t' + str(1.0/(mveraccept[0]))  + '\t' + str((avmverres[0])) +
          '\t' + str(cum2[0])  + '\t' + str(skewmverres[0]) + '\t' + str(kurtmverres[0]))
if (tvi=='on'):
    res.write('\t '+  str(1.0/(mhoraccept[0])) + '\t' + str((avmhorres[0])) + '\t' +
              '\t' + str(cum2hor[0]) +
              '\t' + str(skewmhorres[0]) + '\t' + str(kurtmhorres[0]))

res.write('\n')
for g in range(len(mveraccept)-1):
    res.write( str(layer[g]) + '\t' +  str(1.0/(mveraccept[g+1])) + '\t'+ str((avmverres[g+1])) +
            '\t' + str(cum2[g+1])  +'\t' + str(skewmverres[g+1]) + '\t' + str(kurtmverres[g+1]))
    if (tvi=='on'):
        res.write('\t'+ str(1.0/(mhoraccept[g+1])) + '\t' + str((avmhorres[g+1])) +
            '\t ' + str(cum2hor[g+1]) + '\t' + str(skewmhorres[g+1]) + '\t' + str(kurtmhorres[g+1]))
    res.write('\n')


nrspaces=100
nrspaces2=100
rhoplass=[0]*nrspaces
zrhoplass=[0]*nrspaces2
z2rhoplass=[0]*nrspaces

minrho=-math.log10(maxcon)
maxrho=-math.log10(mincon)
zminrho=-0.5*(zul*zul - math.pow((zul-(layer[2]-layer[1])),2.0))/((layer[2]-layer[1]))
zmaxrho=0.5*(zul*zul - math.pow((zul-(layer[2]-layer[1])),2.0))/((layer[2]-layer[1]))


### WRITING AVERAGE OF AVERAGE RESISITIVIY AND CENTER OF RESISTIVITY TO FILE
for i in range(len(rhointegrated)):
    if (zrhointegrated[i] > zmaxrho or zrhointegrated[i] < zminrho ):
        continue
    rhoplass[int((rhointegrated[i]-minrho)*(nrspaces-1)/(maxrho-minrho))]+=1
    zrhoplass[int((zrhointegrated[i]-zminrho)*(nrspaces-1)/(zmaxrho-zminrho))]+=1

for q in range(nrspaces):
    resres.write(str(minrho + q*(maxrho-minrho)/nrspaces) + '\t' + str(rhoplass[q]) +
                 '\t'+str(zminrho + q*(zmaxrho-zminrho)/nrspaces) + '\t' + str(zrhoplass[q]) + '\n')



### WRITING MARGINAL PROBABILITY DISTRIBUTIONS FOR EACH LAYER TO FILE
for j in range(nrcum+ad):
    cum.write(str(cumint*(j-ad) + 0.5*cumint) + '\t')
    for k in range(len(layer)+1):
        cum.write(str(laycum[k][j]) + '\t')
    cum.write('\n')


if (tvi=='on'):
    for j in range(nrcum+ad):
        cumhor.write(str(cumint*(j-ad) + 0.5*cumint) + '\t')
        for k in range(len(layer)+1):
```

```
                cumhor.write(str(laycumhor[k][j]) + '\t')
            cumhor.write('\n')
    ### END MAIN

################ METHOD TO ALLOCATE STARTMODEL ###################################

def startmodel(mstart, nrzlay,startmod,i):

    mver_old = [mstart[0]]
    if (startmod == 'none'):
        for m in range(nrzlay):
            temp=mstart[1]
            mver_old.append(temp)
    else:
        start=open(startmod, 'r')
        startl=open(startmod, 'r')

        test=start.read()
        lines=startl.readlines()

        mver_old[0]=eval(test.split()[i])
        for l in range(len(lines)-2):
            mver_old.append(eval(test.split()[(l+1)*3 + i]))
  #          print l, (eval(test.split()[(l+1)*3 + i]))

    return mver_old
################### METHOD FOR TAKING A CONDUCTIVITY STEP ########################
def step(bool, max, min,pos_old, maxstep):
    r=random.random()
    deltacon = (math.log10(max)-math.log10(min))*(r-0.5)*maxstep
    while not bool:
        delta = math.pow(10,deltacon)
        pos_new=pos_old*delta
        bool=True
        if (pos_new < (min)):
            bool=False
            deltamin= math.log10(min/pos_old)
            deltacon-=2*deltamin
            deltacon*=-1
        if (pos_new > max):
            bool=False
            deltamax=math.log10(max/pos_old)
            deltacon-=2*deltamax
            deltacon*=-1

    return delta

###################### METHOD FOR UPDATING THE STEP LENGTH #########################
def uppdate_step(accept, reject, step):

    if ((reject - accept) == 0):
        return step

    if (float(accept)/(reject+accept) > 0.5):
        step*=1.5

    if (float(accept)/(accept+reject) < 0.5):
        step/=1.5

    if (step > 1.0):
        step=1.0
```

```
    return step

###################### METOD FOR CALCULATING MISFIT USING ELCARDINAL ############################
def calc_misfit(model,dataobs,alpha,noisefile,frequencies,r0,dr,rl,mver,
                mhor,zlay,tvimod,w,randomSeed):

    filename = 'model' + str(model) + '_rs'+str(randomSeed)+'.nc'
    sr = pyelio.SurveyResult(filename, pyelio.FileOpenMode.READ_ONLY)

    #Choosing the Ex-channal for the test model!
    ex1=sr.getChannel(pyelio.ChannelType.CH_EMF_EX,1)
    if ex1.hasFrequencyResult():
        ex1_freq = ex1.getFrequencyResult()
        freqs = ex1_freq.getFrequencies()
    misf= 0
    maxoffset=[]

    #Calculating misfit: |Ex(offset)-Ex_obs(offset)|^2/(|Ex|^2 + err)
    nopoints=0
    for i in range(len(frequencies)):
        data = list(freqs[i].getComplexSamples())
        n= (1500 - r0)/dr
        n=int(n)
        maxoffset.append(rl - i*500)
        j=(rl -maxoffset[i])/dr
        max=round(j)
        max=0

        for k in range(len(dataobs[i])-(n+max)):
            misf += (abs(data[k+n]-dataobs[i][k+n])*abs(data[k+n]-dataobs[i][k+n]))*(w[i][k+n])

    return misf
#######################METHOD FOR CALCULATING MISFIT USING THE FRECHET DERIVATIVES ################

def  calc_misfit_Frechet(model, dataobs,alpha, noisefile,frequencies, delta,l,tvi,r0
                         ,dr,mver,mhor,zlay,tvimod,w,randomSeed):
    r=open('f' + str(model) + '_rs'+str(randomSeed), 'rb')

    dat2=[]
    counter=0
    sizef=struct.calcsize('f')
    while 1:
        data2=r.read(sizef)
        if (data2== ''):
            break

        if (counter%2==0):
            num=struct.unpack('f', data2)
            num=num[0]*complex(1,0)

        if (counter%2==1):
            numj=struct.unpack('f', data2)
            num += numj[0]*complex(0,1)
            dat2.append(num)

        counter+=1

    sr = pyelio.SurveyResult('model' + str(model) +
                             '_rs'+str(randomSeed)+ '.nc', pyelio.FileOpenMode.READ_ONLY)

    #Choosing the Ex-channal for the test model!
    ex1=sr.getChannel(pyelio.ChannelType.CH_EMF_EX,1)
```

```
    if ex1.hasFrequencyResult():
        ex1_freq = ex1.getFrequencyResult()
        freqs = ex1_freq.getFrequencies()
    misf= 0
    nofreqs=len(frequencies)
    nopoints=0
    for i in range(len(frequencies)):
        data = list(freqs[i].getComplexSamples())
        max=0
        n= (1500 - r0)/dr
        n=int(n)

        ### Estimating new data for Ex
        if(tvi=='on'):
            for k in range(len(data)-(n+max)):
                add=0
                for l in range (len(delta)/2):
                    lay=l+(len(delta)/2)
                    add+= (dat2[k+n+(lay*nofreqs + i)*len(data)]*delta[l])
                    add+= (dat2[k+n+(l*nofreqs + i)*len(data)]*delta[lay])
                data[k+n] += add
                ### Calculating data misfit
                misf += (abs(data[k+n]-dataobs[i][k+n])*abs(data[k+n]-dataobs[i][k+n]))*(w[i][k+n])

        else:
            for k in range(len(data)-(n+max)):
                add=0
                for l in range (len(delta)):
                    lay=l+(len(delta))
                    add+= (dat2[k+n+(l*nofreqs + i)*len(data)]*delta[l])
                data[k+n] += add
                ### Calculating data misfit
                misf += (abs(data[k+n]-dataobs[i][k+n])*abs(data[k+n]-dataobs[i][k+n]))*(w[i][k+n])


    return misf

###################### METHOD FOR CALCULATING MISFIT BASED ON REGULARIZATION ##########################
def calc_misf_reg(frequencies,mver, mhor,zlay,tvimod):
    s=2
    reg2=0
    reg=0

    for r in range(len(mver)-s):
        reg+=math.pow((math.log10(mver[r+s])-math.log10(mver[r+s-1])),2)

    if (tvimod==2):
        for r in range(len(mver)-s):
            reg2+=math.pow((math.log10(mhor[r+s])-math.log10(mhor[r+s-1])),2)

            if (mhor[r+s-1]< mver[r+s-1]):
                reg2+=10000000*math.pow((math.log10(mhor[r+s-1]) - math.log10(mver[r+s-1])),2)

            else:
                if (abs(mhor[r+s-1])/(mver[r+s-1]) > 3.0):
                    reg2+=5*abs(math.pow((math.log10(mver[r+s-1])-math.log10(mhor[r+s-1])),2))

    reg+=reg2
    if tvimod==2:
        reg/=2.0

    return reg
```

```
############################ METHOD FOR MAKING COMMAND ################################################

def mkcommand(model, mver, mhor, layers, zr, tvi, acur, frequencies ,r0,dr,rl,randomSeed):
    filename= 'model' + str(model) + '_rs'+str(randomSeed)+'.nc'

    if (tvi=='off'):
        mhor=mver
    if (len(layers)!= len(mver)-1) :
        print 'Error: Each layer needs a spesified value for the resistivity/conductivity'
        print len(layers), len(mver)

    zs= zr - 30
    frex='f'+str(model) + '_rs'+str(randomSeed)

    command = binDir + 'elcardinal '
    command += 'ofel=' + filename
    command += ' par=' + 'con'
    command += ' acur='+str(acur)
    command += ' comp=ex'
    command += ' method=digf'
    if (model !=0):
        command += ' frex=' + frex
    command += ' zlay='
    for i in range(len(layers)-1):
        command+= str(layers[i]) + ','
    command += str(layers[len(layers)-1])
    command += ' freq=' + str(frequencies[0])
    for i in range(len(frequencies) - 1):
        command += ',' + str(frequencies[i+1])
    command += ' tvi=' + tvi
    command += ' mver='
    for i in range(len(mver)-1):
        command +=  str(mver[i]) + ','
    command += str(mver[len(mver) - 1])
    command += ' mhor='
    for i in range(len(mhor)-1):
        command +=  str(mhor[i]) + ','
    command += str(mhor[len(mhor) - 1])
    command += ' zr='+ str(zr)
    command += ' zs='+ str(zs)
    command += ' r0='+ str(r0)
    command += ' dr='+ str(dr)
    command += ' rl='+ str(rl)

    return command


#################################### LogClass ##################################
class LogClass:
    progName = ""
    verbose  = 0

    def __init__(self,progName,verbose):
        self.progName = progName
        self.verbose  = verbose

    def write(self,message,funcName=""):
        if (self.verbose > 0):
            print "# "+self.progName+": "+funcName+" "+message

    def writeln(self,message,funcName=""):
```

```
        if (self.verbose > 0):
            print
            print "# "+self.progName+": "+funcName+" "+message

    def WRITE(self,message,funcName=""):
        print "# "+self.progName+": "+funcName+" "+message

    def WRITELN(self,message,funcName=""):
        print
        print "# "+self.progName+": "+funcName+" "+message

    def eWrite(self,message,funcName=""):
        print
        print "# "+self.progName+": "+funcName+": ERROR: "+message
        sys.exit(-1)

    def printOptions(self,argDict):
        for key in argDict:
            self.write("     "+key+"="+argDict[key])
############################## END LogClass #######################################


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

### STARTING MAIN ######
if __name__=="__main__":
    main(sys.argv)
```

B.4 Code for collecting the results when doing parallel computing

```
#! /usr/bin/env python
import sys
import os.path
import struct
import math
#Set dir for local modules.
binDir = os.path.dirname(sys.argv[0])
binDir = os.path.abspath(binDir)+'/'
sys.path.append(binDir) # Where to find the following modules:

import arguments


def main(argv):
    optionList = [
        'files=resultDir/result.',              'The file containing observed data',
        'paraJobs=1',                           'Number of parallel jobs',
        'result=resultDir/collectedFiles/result','The name of the resultfile. The resulting name will be resultfilename
        'tvi=off',                              'tvi=on or tvi=off',
        'maxcon=10.0',                          'max condutctivity',
        'mincon=0.01',                          'min conductivity',
        ]


    nrcum    = 40
    arg      = arguments.arguments(optionList,argv)
```

```
vars     = arg.getargs()
files    = vars['files']
paraJobs = int(vars['paraJobs'])
result   = vars['result']
tvi      = vars['tvi']
maxcon   = float(vars['maxcon'])
mincon   = float(vars['mincon'])


ad=-round(math.log10(1.0/maxcon)/math.log10(1.0/mincon)*nrcum)
ad=int(ad)

newmax  = ad*math.log10(1.0/mincon)/nrcum
maxcon  = math.pow(10,newmax)
tvimod  = 1

if (tvi=='off'):
    tvimod=0
resvarnames        = []
resvarnamesres     = []
resvarnameslst     = []
resvarnameslsthor  = []

depth   = [0]
rhonr   = [0]
zrhonr  = [0]
rho     = [0]
zrho    = [0]

mveravnew    = [0]
stdnew       = [0]
mhoravnew    = [0]
stdnewhor    = [0]
rhonrnew     = [0]
zrhonrnew    = [0]
modelsnew    = []
modelsnewhor = []
ncorr        = [0]


for i in range(paraJobs):
    resvarnames.append(files + str(i) + '.dat')
    resvarnamesres.append(files + str(i) + 'res.dat')
    resvarnameslst.append(files + str(i) + 'lst.dat')

    if (tvi=='on'):
        resvarnameslsthor.append(files + str(i) + 'lsthor.dat')

for i in range(paraJobs):

    mverav = [0]
    std    = [0]
    mhorav = [0]
    stdhor = [0]
    ncorr  = [0]
    name   = resvarnames[i]
    nameres= resvarnamesres[i]
    namelst= resvarnameslst[i]

    resres      =open(nameres, 'r')
    res         =open(name,'r')
    reslines    =open(name,'r')
```

```
reslines2   =open(nameres,'r')
reslayers   =open(namelst,'r')

if (tvi=='on'):
    namelsthor  = resvarnameslsthor[i]
    reslayershor= open(namelsthor,'r')
    layershor   = reslayershor.readlines()


test        = res.read()
testres     = resres.read()
lines       = reslines.readlines()
lines2      = reslines2.readlines()
layers      = reslayers.readlines()


mverav[0]   = eval(test.split()[2])
std[0]      = eval(test.split()[3])
if (tvi=='on'):
    mhorav[0] = eval(test.split()[7])
    stdhor[0] = eval(test.split()[8])

rhonr[0]    = eval(testres.split()[1])
zrhonr[0]   = eval(testres.split()[3])
models      = []
modelshor   = []

for j in range(len(layers)):
    models.append(eval(layers[j]))
    if (tvi=='on'):
        modelshor.append(eval(layershor[j]))

if (i==0):
    for j in range(len(layers)):
        modelsnew.append(eval(layers[j]))
        if(tvi=='on'):
            modelsnewhor.append(eval(layershor[j]))
else:
    for k in range(len(layers)):
        modelsnew[k]+=(models[k])
        if(tvi=='on'):
            modelsnewhor[k]+=modelshor[k]



if (i==0):
    mveravnew[0] = mverav[0]
    stdnew[0]    = std[0]
    mhoravnew[0] = mhorav[0]
    stdnewhor[0] = stdhor[0]
    rhonrnew[0]  = rhonr[0]
    zrhonrnew[0] = zrhonr[0]
    depth[0]     = eval(test.split()[0])
    rho[0]       = eval(testres.split()[0])
    zrho[0]      = eval(testres.split()[2])


    for l in range(len(lines)-1):
        depth.append(eval(test.split()[(l+1)*(6+(tvimod*5)) + 0]))
    for l in range(len(lines2)-1):
        rho.append(eval(testres.split()[4*l]))
        zrho.append(eval(testres.split()[4*l + 2]))
```

```
    for l in range(len(lines)-1):
        mverav.append(eval(test.split()[(l+1)*(6+(tvimod*5)) + 2]))
        std.append(eval(test.split()[(l+1)*(6+(tvimod*5)) + 3]))
        if (tvi=='on'):
            mhorav.append(eval(test.split()[(l+1)*(6+(tvimod*5)) + 7]))
            stdhor.append(eval(test.split()[(l+1)*(6+(tvimod*5)) + 8]))

        if (i ==0):
            mveravnew.append(mverav[l+1])
            stdnew.append(std[l+1])
            if (tvi=='on'):
                mhoravnew.append(mhorav[l+1])
                stdnewhor.append(stdhor[l+1])
        else:
            mveravnew[l]+= mverav[l]
            stdnew[l]+=std[l]
            if(tvi=='on'):
                mhoravnew[l] += mhorav[l]
                stdnewhor[l] += stdhor[l]

    for l in range(len(lines2)-1):
        rhonr.append(eval(testres.split()[4*l + 1] ))
        zrhonr.append(eval(testres.split()[4*l + 3]))
        if (i ==0):
            rhonrnew.append(rhonr[l+1])
            zrhonrnew.append(zrhonr[l+1])
        else:
            rhonrnew[l]  += rhonr[l]
            zrhonrnew[l] += zrhonr[l]

    if (i!=0):
        mveravnew[len(mveravnew)-1] += mverav[len(mverav)-1]
        stdnew[len(mveravnew)-1]    += std[len(mveravnew)-1]
        rhonrnew[len(rhonrnew)-1]   += rhonr[len(rhonrnew)-1]
        zrhonrnew[len(zrhonrnew)-1] += zrhonr[len(zrhonrnew)-1]

        if (tvi == 'on'):
            mhoravnew[len(mhoravnew)-1] += mhorav[len(mhorav)-1]
            stdnewhor[len(mhoravnew)-1] += stdhor[len(mhoravnew)-1]

for k in range(len(mveravnew)):
    stdnew[k]   /= paraJobs
    mveravnew[k]/= paraJobs
    stdnew[k]    = math.sqrt(stdnew[k]-(mveravnew[k]*mveravnew[k]))
    if (tvi == 'on'):
        stdnewhor[k] /= paraJobs
        mhoravnew[k] /= paraJobs
        stdnewhor[k]  = math.sqrt(stdnewhor[k]-(mhoravnew[k]*mhoravnew[k]))
sumver=[]
sumhor=[]
corr=[]
corrhor=[]

for i in range(len(modelsnew)):
    corr.append([0]*(len(modelsnew)))
    corrhor.append([0]*(len(modelsnew)))

for k in range(len(modelsnew)):
    for l in range(len(modelsnew)):
        a=0
        b=0
```

```
        for i in range(len(modelsnew[0])):

            a+=(math.log10(modelsnew[k][i])-(mveravnew[k+1]))*(math.log10(modelsnew[l][i]) - (mveravnew[l+1]))
            if (tvi =='on'):
                b+=(math.log10(modelsnewhor[k][i])-(mhoravnew[k+1]))*(math.log10(modelsnewhor[l][i])
  - (mhoravnew[l+1]))

        a /= (len(modelsnew[0]))
        b /= (len(modelsnew[0]))

        if (k==l):
            sumver.append(math.sqrt(a))
            sumhor.append(math.sqrt(b))

        corr[k][l]    = a
        corrhor[k][l] = b

for k in range(len(modelsnew)):
    for l in range(len(modelsnew)):
        corr[k][l]/=(sumver[k]*sumver[l])
        if(tvi=='on'):
            corrhor[k][l]/=(sumhor[k]*sumhor[l])
    print corr[k][k], k


# THE CUMULATIVE FILES
logres         = [0]
logreshor      = [0]
resvarnames    = []
resvarnameshor = []
laycumnew      = []
laycumnewhor   = []
for k in range(len(depth)):
    laycumnew.append([0]*(nrcum+ad))
    if (tvi=='on'):
        laycumnewhor.append([0]*(nrcum+ad))

for i in range(paraJobs):
    resvarnames.append(files + str(i) + 'cum.dat')
    if (tvi=='on'):
        resvarnameshor.append(files + str(i) + 'cumhor.dat')
for i in range(paraJobs):
    name           = resvarnames[i]
    rescum         = open(name,'r')
    reslinescum    = open(name,'r')
    test           = rescum.read()
    lines          = reslinescum.readlines()
    if (tvi=='on'):
        namehor        = resvarnameshor[i]
        rescumhor      = open(namehor, 'r')
        testhor        = rescumhor.read()
        reslinescumhor = open(namehor, 'r')
        lineshor       = reslinescumhor.readlines()

    if (i==0):
        logres[0] = eval(test.split()[0])
        for l in range(len(lines)-1):
            logres.append(eval(test.split()[(l+1)*(len(depth)+1) + 0]))
        if(tvi=='on'):
            logreshor[0] =eval(testhor.split()[0])
            for l in range(len(lines)-1):
                logreshor.append(eval(testhor.split()[(l+1)*(len(depth)+1) + 0]))
```

```
        for l in range(len(depth)):
            for j in range(len(logres)):
                laycumnew[l][j]+=(eval(test.split()[(l+1 + j*(len(depth)+1))]))

        if (tvi=='on'):
            for l in range(len(depth)):
                for j in range(len(logreshor)):
                    laycumnewhor[l][j]+=(eval(testhor.split()[(l+1 + j*(len(depth)+1))]))




    rescollect       = open(result, 'w')
    rescollectcum    = open(result + 'cum', 'w')
    rescollectcorr   = open(result + 'corr', 'w')
    rescollectres    = open(result + 'res', 'w')
    rescollectlst    = open(result + 'lst', 'w')
    resultname6      = result + 'matrix'
    rescollectmatrix = open(resultname6, 'w')
    if (tvi=='on'):
        rescollecthor    = open(result + 'hor', 'w')
        rescollectcumhor = open(result + 'cumhor', 'w')
        rescollectlsthor = open(result + 'lsthor', 'w')

    print 'WRITING TO FILES'
    for z in range(len(modelsnew[0])):
        for w in range(len(modelsnew)):
            rescollectlst.write(str(modelsnew[w][z]) + ',\t')
        rescollectlst.write(str(modelsnew[w][z]))
        rescollectlst.write('\n')

### WRITING CORRELATION MATRIX TO FILE

    for k in range (len(modelsnew)-1):
        rescollectmatrix.write(str(depth[k+1]) + '\t')
        for l in range ((len(modelsnew)-1)+tvimod*(len(modelsnew)-1)):
            if (l < len(modelsnew)-1):
                rescollectmatrix.write(str(corr[k+1][l+1]) + '\t')
            else:
                rescollectmatrix.write(str(corrhor[k+1][l-len(modelsnew)]) + '\t')
        rescollectmatrix.write('\n')

    if (tvi=='on'):
        for z in range(len(modelsnew[0])):
            for w in range(len(modelsnew)-1):
                rescollectlsthor.write(str(modelsnewhor[w][z]) + ',\t')
            rescollectlsthor.write(str(modelsnewhor[w][z]))
            rescollectlsthor.write('\n')


    rescollectcorr.write(str(0.0)+ '\t' + str(0.0)+ '\n')

    for k in range (len(modelsnew)-1):
        if (tvi=='on'):
            rescollectcorr.write(str(depth[k+1]) + '\t' +str(corr[k][k+1])  + '\t' + str(corrhor[k][k+1]))
        else:
            rescollectcorr.write(str(depth[k+1]) + '\t' +str(corr[k][k+1]))
        rescollectcorr.write('\n')


    for w in range(len(rhonrnew)):
```

```
        rescollectres.write(str(rho[w]) + '\t' + str(rhonrnew[w]) + '\t' + str(zrho[w]) + '\t'
                  + str(zrhonrnew[w]) + '\n')

    for w in range(len(mveravnew)):
        rescollect.write(str(depth[w]) + '\t' + str(mveravnew[w]) + '\t' + str(stdnew[w]) + '\n')
        if (tvi=='on'):
            rescollecthor.write(str(depth[w]) + '\t' + str(mhoravnew[w]) + '\t' + str(stdnewhor[w]) + '\n')

    for j in range(nrcum+ad):
        rescollectcum.write(str(logres[j]) + '\t')
        for k in range(len(depth)):
            rescollectcum.write(str(laycumnew[k][j]) + '\t')
        rescollectcum.write('\n')

    if (tvi=='on'):
        for j in range(nrcum+ad):
            rescollectcumhor.write(str(logreshor[j]) + '\t')
            for k in range(len(depth)):
                rescollectcumhor.write(str(laycumnewhor[k][j]) + '\t')
            rescollectcumhor.write('\n')



    rescollect.close()
    rescollectcum.close()

    rescollectcorr.close()
    rescollectres.close()
    rescollectlst.close()
    rescollectmatrix.close()

    if (tvi=='on'):
        rescollectcum.close()
        rescollecthor.close()
        rescollectlsthor.close()
    print 'FINITO'

######################################################################
if __name__=="__main__":
    main(sys.argv)
######################################################################
```

B.5 Calculating probabilities

```
#! /usr/bin/env python
import sys
import os.path
import struct
import math
#Set dir for local modules.
binDir = os.path.dirname(sys.argv[0])
binDir = os.path.abspath(binDir)+'/'
sys.path.append(binDir) # Where to find the following modules:

import arguments
progName = "calcProb.py"
infostring = """
NAME:calcProb.py
The script reads in the file containing all models that are sampled from the a posteriori
probability distribution. It then counts the number of models that has values consistent
```

```
with the ones given by the user and devides this number by the total number of models.
The models that fit the criteria are all models that have an average resistivity which is
< *maxvalue* and > *minvalue* in the depth interval from *mindepth* to *maxdepth*.

If *checkall=off* then you check only for the given depth interval.
If *checkall=on* then the script first calculates the probability that
the models have an average resistivity which is < *maxvalue* and > *minvalue*
in the depth interval from seafloor depth to seafloor depth + *interval*.
Next, it calculates the probability that the models have an average resistivity
which is < *maxvalue* and > *minvalue*
in the depth interval from (seafloor depth + layerthickness) to
(seafloor depth + layerticknes + *interval*) etc.

"""

def main(argv):
    if('-help' in argv):
        print infostring
        sys.exit(0)
    optionList = [
        'readfile=result',     'Filename for the collected resultfiles',
        'mindepth=2300.0',     'lower limit',
        'maxdepth=2400.0',     'upper limit for testing hypotesis',
        'maxvalue=5000.0',     'How many models smaller than this value',
        'minvalue=1000.0',     'How many models greater than this value',
        'checkall=off',        'Check more that one value',
        'interval=100.0',      'interval spacing: must be a least 100.0 m',
        ]



    arg       = arguments.arguments(optionList,argv)
    vars      = arg.getargs()
    readf     = vars['readfile']
    readfile  = vars['readfile']+'lst'
    maxdepth  = float(vars['maxdepth'])
    mindepth  = float(vars['mindepth'])
    minvalue  = float(vars['minvalue'])
    maxvalue  = float(vars['maxvalue'])
    checkall  = vars['checkall']
    interval  = float(vars['interval'])

    if (maxdepth <= mindepth):
        print 'Error: maxdepth <= mindepth'
        sys.exit(0)

    res         = open(readf,'r')
    reslines    = open(readf,'r')
    test        = res.read()
    testlines   = reslines.readlines()

    depth=[]
    first=True
    firstmax=True

    if (checkall=='off'):
        for k in range(len(testlines)):
            depth.append(eval(test.split()[3*k]))

            if (depth[k]>=mindepth and first):
                lowerlayer=k-1
                first=False
```

```
        if (depth[k]>=maxdepth and firstmax):
            upperlayer=k-1
            firstmax=False


    file =open(readfile, 'r')
    mod = file.readlines()
    models=[]

    probmodel=0
    totmodel=0

    for i in range(len(mod)):

        models.append(eval(mod[i]))
        totmodel+=1
        sum=0
        for l in range(lowerlayer, upperlayer):
            sum+=(models[i][l])*(depth[l+2]-depth[l+1])

        if (sum >= minvalue and sum <=maxvalue ):
            probmodel+=1



    probmodel/=float(totmodel)
    print probmodel
    sys.exit(0)
else:
    resprob=open('resprob', 'w')
    for k in range(len(testlines)):
        depth.append(eval(test.split()[3*k]))

    models=[]
    totmodel=0
    for k in range(len(testlines)-int(interval/100) -1):
        lowerlayer  = k
        upperlayer  = int((interval/100)) + k

        file = open(readfile, 'r')
        mod  = file.readlines()

        probmodel = 0

        for i in range(len(mod)):
            if (k==0):
                models.append(eval(mod[i]))
                totmodel+=1

            sum=0
            for l in range(lowerlayer, upperlayer):

                sum+=(models[i][l])*(depth[l+2]-depth[l+1])
            if (sum >= minvalue and sum<= maxvalue):
                probmodel+=1

        probmodel/=float(totmodel)
        print probmodel
        resprob.write(str((depth[lowerlayer+1]+ depth[upperlayer+1])/2.0 ) + '\t'
                      + str(probmodel) + '\n')

resprob.close()
```

```
#####################################################################
if __name__=="__main__":
    main(sys.argv)
#####################################################################
```

B.6 Simulated Annealing to find start model

```
#! /usr/bin/env python
import sys
import os
import os.path

#Set dir for local modules.
#binDir = os.path.dirname(sys.argv[0])
#binDir = os.path.abspath(binDir)+'/'
#sys.path.append(binDir) # Where to find the following modules:

import shutil
import loghelper
import arguments
import pyelio #Needed to read files of type <filename>.nc
import random
import math
import struct
import pynavutil
#import decimal


progName = "sa.py"
infostring = """
NAME:sa.py
This script performes a simulated annealing search for the model of best data fit with the
observed data from ifel.
The Frechet derivatives are used to estimate the E_x data for each point.

The output file with the name spesified by the user contain, consists of three columns:

#########################################################################
#|Depth (m) | Vertical conductivity | Horizontal conductivity |     #
#########################################################################

""" #end multiline string


def main(argv):
    if('-help' in argv):
        print infostring
        sys.exit(0)
    optionList = [

        'ifel=Tr_007_Data.nc',          'The file containing observed data',
        'ofel=starthei',                'The filename where the result model is saved',
        'waterdepth=332.0',             'Depth to seafloor',
        'maxdepth=4332.0',              'Depth of the deepest layer boundary',
        'nrzlay=41',                    'The number of layers below the seafloor (waterlayer not included
        'maxcon=10.0',                  'The maximum conductivity',
        'mincon=0.01',                  'The minimum conductivity',
        'mverstart=3.69,0.25',          'The vertical water conductivity',
        'mhorstart=3.69,0.50',          'The horizontal water conductivity',
```

```
    'tvi=off',                      'tvi=off forces horizontal and vertical conductivity equal.Option tvi=on',
    'skip=0',                       'Number of models to skip in the beginning of the Monte Carlo simulation',
    'alpha=0.05',                   'The apha in the error in Ex',
    'noisefile=Tr_007_Noise.nc',    'The input noise file. If noisefile=none the noisefloor value is used',
    'noisefloor=-14',               'The noise level. 10^noise'  ,
    'acur=4',                       'accuracy in integration',
    'it=5',                         'Max iterations in the simmulated annealing scheme',
    ]


arg = arguments.arguments(optionList,argv)
vars = arg.getargs()


obsfile    = vars['ifel']
ofel       = vars['ofel']
noisefile  = vars['noisefile']
waterdepth = float(vars['waterdepth'])
maxcon     = float(vars['maxcon'])
mincon     = float(vars['mincon'])
nrzlay     = int(vars['nrzlay'])
maxdepth   = float(vars['maxdepth'])
spacezlay  = int(vars['nrzlay'])
it         = int(vars['it'])
mverstart  = eval(vars['mverstart'])
mhorstart  = eval(vars['mhorstart'])
acur       = vars['acur']
tvi        = vars['tvi']
skip       = int(vars['skip'])
alpha      = float(vars['alpha'])
noisefloor = float(vars['noisefloor'])

if (noisefile == 'none'):
    noise = math.pow(10,noisefloor)
    noise_dat = [math.pow(10,noisefloor)]

mverwater=mverstart[0]
mhorwater=mhorstart[0]

par         = 'con'
frequencies = []

if (noisefile!= 'none'):
    noise=pyelio.SurveyResult(noisefile, pyelio.FileOpenMode.READ_ONLY)
    ex1noise=noise.getChannel(pyelio.ChannelType.CH_EMF_EX,1)
    if ex1noise.hasFrequencyResult() :
        noisef = ex1noise.getFrequencyResult().getFrequencies()

obsr = pyelio.SurveyResult(obsfile, pyelio.FileOpenMode.READ_ONLY)

#Ex-kanalen
ex1obs=obsr.getChannel(pyelio.ChannelType.CH_EMF_EX,1)
if ex1obs.hasFrequencyResult() :
    freqsobs = ex1obs.getFrequencyResult().getFrequencies()
    for f in freqsobs :
        frequencies.append(f.getValue())
        print f.getValue()


w          = []
data_obs   = []
```

```
    noise_data = []

    ###READING DATA FROM TRUE MODEL AND CALCULATING WEIGTHS: 1/(alpha^2*Eo^2 + noise^2)
    ###IF noise to signal ratio less than 10 then the weight=0

    for i in range(len(frequencies)):
        if (noisefile !='none'):
            noise_data.append(list(noisef[i].getComplexSamples()))
        data_obs.append(list(freqsobs[i].getComplexSamples()))

    for i in range(len(frequencies)):
        if (noisefile == 'none'):
            noise_data.append(noise_dat)

        w.append([0.0]*len(data_obs[i]))
        for k in range(len(data_obs[i])):
            if len(noise_data[i])>1:
                noise=abs(noise_data[i][k])
            datobs=abs(data_obs[i][k])
            weight= 1.0/((alpha*alpha)*(datobs*datobs) + noise*noise)
            w[i][k]=(weight)
            ratio=abs(datobs)/(abs(noise))
            if (ratio < 10):
                w[i][k]=0



    nav_x = obsr.getChannel(pyelio.ChannelType.CH_SOURCE_X, 1)
    nav_y = obsr.getCha
\par
 \end{center}
nnel(pyelio.ChannelType.CH_SOURCE_Y, 1)
    sx = nav_x.getFrequencyResult().getFrequencies()[0].getSamples()
    sy = nav_y.getFrequencyResult().getFrequencies()[0].getSamples()

    # Receiver coordinates
    rec_x = obsr.prop_receiver_x
    rec_y = obsr.prop_receiver_y

    # using navutils offset calculatro
    offsets = pynavutil.calculate_offsets(sx, sy, rec_x, rec_y)

    rl = offsets[len(offsets)-1]
    dr = offsets[1]-offsets[0]
    r0 = offsets[0]

    layer=[]
    for l in range(nrzlay):
        ld=waterdepth + l*(maxdepth-waterdepth)/(nrzlay-1)
        layer.append(ld)


    tvimod=1
    if (tvi =='on'):
        tvimod=2


    maxstep  = 0.01
    start    = open(ofel, 'w')
    #INITIAL MODEL
    count    = 0
    nrmoves  = 0
```

```
mver_old=startmodel(mverstart, nrzlay)
mhor_old=startmodel(mhorstart,nrzlay)

if (tvimod==1):
    for i in range(len(mver_old)):
        mhor_old[i]=mver_old[i]


zr  = layer[0]

cmd          = mkcommand(1, mver_old, mhor_old, layer, zr, tvi,acur, frequencies, r0,dr,rl)
loghelper.runCommand(cmd)
misf_old     = calc_misfit(1,data_obs,alpha,noise,frequencies,r0,dr,rl,mver_old,mhor_old,layer,tvimod,w)
misf_old_reg = calc_misf_reg(mver_old, mhor_old,layer,tvimod)


probability_old = math.exp(-misf_old)

temp        = 1.0
delta       = [0]*(tvimod*len(mver_old)-tvimod)
mv_good     = [0]*len(mver_old)
mh_good     = [0]*len(mhor_old)
acceptround = 0

cmd          = mkcommand(1, mver_old, mhor_old, layer, zr, tvi,acur, frequencies, r0,dr,rl)
loghelper.runCommand(cmd)
misf_old     = calc_misfit(1,data_obs,alpha,noise,frequencies,r0,dr,rl,mver_old,mhor_old,layer,tvimod,w)
misf_old_reg = calc_misf_reg(mver_old, mhor_old,layer,tvimod)
while (count < it or misf_old < 0.01):
    temp        /= 1.001
    count       += 1
    cont         = False

    for i in range(len(mver_old)):
        mv_good[i] = mver_old[i]
        mh_good[i] = mhor_old[i]
    misf_good      = misf_old
    misf_reg_good  = misf_old_reg
    delta = []
    delta = [0]*(tvimod*len(mver_old)-tvimod)
    print mver_old
    print 'misf_old', misf_old
    for l in range(len(layer)*tvimod):

        mv=[]
        mh=[]
        for i in range(len(mver_old)):
            mv.append(mver_old[i])
            mh.append(mhor_old[i])

        inrange=False

        ### TAKING A STEP
        if (l < len(layer)):
            deltacon = step(inrange, maxcon, mincon, mver_old[l+1], maxstep)
            mv[l+1] *= deltacon
            delta[l] = mv[l+1]-mver_old[l+1]


        else:
            deltacon              = step(inrange,maxcon,mincon,mhor_old[l -len(layer)+1],maxstep)
```

```
        mh[l - len(layer) +1]*= deltacon
        delta[l]              = mh[l-len(layer)+1]-mhor_old[l-len(layer)+1]

    misfreg=calc_misf_reg(mv, mh,layer,tvimod)
    r=random.random()

    ### ACCEPTANCE TEST BASED ON REGULARIZATION
    if (misfreg < misf_old_reg or r < math.exp((-misfreg + misf_old_reg)/temp)):
        misf = calc_misfit_Frechet(1, data_obs,alpha, noise,frequencies,delta,0,
                   tvi,r0,dr,mv, mh,layer,tvimod,w)
        r=random.random()

        ###ACCEPTANCE TEST BASED ON DATA
        if (misf < misf_old or r < math.exp((-misf + misf_old)/temp)):
            acceptround+=1
            misf_old=misf
            misf_old_reg=misfreg
            print 'ACCEPT'
            if (tvi=='on'):
                if (l < len(layer)):
                    mver_old[l+1]=mv[l+1]
                else:
                    mhor_old[l-len(layer)+1]=mh[l-len(layer)+1]

            else:
                if (l < len(layer)):
                    mver_old[l+1]=mv[l+1]
                    mhor_old[l+1]=mv[l+1]

        else:
            print 'REJECT'
            if (l < len(layer)):
                delta[l]=0
            else:
                delta[l]=0
    else:
        print 'REJECT REG'
        if (l < len(layer)):
            delta[l]=0
        else:
            delta[l]=0

### CHECKING THE ERROR MADE IN THE MISFIT BY USING ELCARDINAL
### IF THE ERROR HIGHER THAN 10
### PARAMETERS AND TRY AGAIN.


cmd      = mkcommand(1, mver_old, mhor_old, layer, zr, tvi,acur, frequencies, r0,dr,rl)
loghelper.runCommand(cmd)
misfelc  = calc_misfit(1,data_obs,alpha,noise,frequencies,r0,dr,rl,mver_old,mhor_old, layer,tvimod,
relerr2  = abs((misf_old+misf_old_reg) -(misfelc+misf_old_reg))/(misfelc+misf_old_reg)
delta    = [0]*(tvimod*len(mver_old)-tvimod)
misf_old = misfelc

print '*****relerr ****', relerr2
if (relerr2 > 0.1):
    for i in range(len(mver_old)):
        mver_old[i] = mv_good[i]
        mhor_old[i] = mh_good[i]

    misf_old=misf_good
    misf_reg=misf_reg_good
```

```
            cont=True

        if (cont==True):
            cont=False
            print 'CONTINUE'
            continue

    start.write('0.0' + '\t'+str(mver_old[0]) + '\t' + str(mhor_old[0]) + '\n')
    for i in range(len(mver_old)-1):
        start.write(str(layer[i]) + '\t'+str(mver_old[i+1]) + '\t' + str(mhor_old[i+1]) + '\n')

    start.write('\n')

#############################################################################################################
def startmodel(mstart, nrzlay):

    mver_old = [mstart[0]]

    for m in range(nrzlay):
        temp=mstart[1]

        mver_old.append(temp)

    return mver_old

#############################################################################################################
def step(bool, max, min,pos_old, maxstep):
    r        = random.random()
    deltacon = (math.log10(max)-math.log10(min))*(r-0.5)*maxstep
    while not bool:
        delta   = math.pow(10,deltacon)
        pos_new = pos_old*delta
        bool    = True
        if (pos_new < (min)):
            bool      = False
            deltamin  = math.log10(min/pos_old)
            deltacon -= 2*deltamin
            deltacon *= -1
        if (pos_new > max):
            bool     = False
            deltamax = math.log10(max/pos_old)
            deltacon -= 2*deltamax
            deltacon *= -1

    return delta

#############################################################################################################
def  calc_misfit_Frechet(model, dataobs,alpha, noisefile,frequencies,
                         delta,l,tvi,r0,dr,mver,mhor,zlay,tvimod,w):
    r=open('f' + str(model), 'rb')
   # print "READING FRECHET FILE", 'f' + str(model)
    dat2   = []
    counter= 0
    sizef  = struct.calcsize('f')
    while 1:
        data2=r.read(sizef)

        if (data2== ''):
            break

        if (counter%2==0):
            num = struct.unpack('f', data2)
```

```
            num = num[0]*complex(1,0)

        if (counter%2==1):
            numj = struct.unpack('f', data2)
            num += numj[0]*complex(0,1)
            dat2.append(num)

        counter+=1

    sr = pyelio.SurveyResult('model' + str(model) + '.nc', pyelio.FileOpenMode.READ_ONLY)
    #Choosing the Ex-channal for the test model!
    ex1=sr.getChannel(pyelio.ChannelType.CH_EMF_EX,1)
    if ex1.hasFrequencyResult():
        ex1_freq = ex1.getFrequencyResult()
        freqs    = ex1_freq.getFrequencies()
    misf     = 0
    nofreqs  = len(frequencies)
    nopoints = 0
    for i in range(len(frequencies)):
        data = list(freqs[i].getComplexSamples())
        max  =0
        n    = (1500 - r0)/dr
        n    = int(n)

        ### Estimating new data for Ex
        if(tvi=='on'):
            for k in range(len(data)-(n+max)):
                add = 0
                for l in range (len(delta)/2):
                    lay  = l+(len(delta)/2)
                    add += (dat2[k+n+(lay*nofreqs + i)*len(data)]*delta[l])
                    add += (dat2[k+n+(l*nofreqs + i)*len(data)]*delta[lay])

                data[k+n] += add
                ### Calculating misf
                misf += (abs(data[k+n]-dataobs[i][k+n])*abs(data[k+n]-dataobs[i][k+n]))*(w[i][k+n])


        else:
            for k in range(len(data)-(n+max)):
                add=0
                for l in range (len(delta)):
                    lay  = l+(len(delta))
                    add += (dat2[k+n+(l*nofreqs + i)*len(data)]*delta[l])

                data[k+n] += add
                misf      += (abs(data[k+n]-dataobs[i][k+n])*abs(data[k+n]-dataobs[i][k+n]))*(w[i][k+n])


    return misf

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

def calc_misfit(model,dataobs,alpha,noisefile,frequencies,r0,dr,rl,mver,mhor,zlay,tvimod,w):

    filename = 'model' + str(model) + '.nc'
    sr       = pyelio.SurveyResult(filename, pyelio.FileOpenMode.READ_ONLY)
    #Choosing the Ex-channal for the test model!
    ex1      = sr.getChannel(pyelio.ChannelType.CH_EMF_EX,1)
    if ex1.hasFrequencyResult():
        ex1_freq = ex1.getFrequencyResult()
        freqs    = ex1_freq.getFrequencies()
```

```
    misf      = 0
    maxoffset =[]

    #Calculating misfit: |Ex(offset)-Ex_obs(offset)|^2/(|Ex|^2 + err)
    nopoints=0
    for i in range(len(frequencies)):
        data = list(freqs[i].getComplexSamples())
        n    = (1500 - r0)/dr
        n    = int(n)
        maxoffset.append(rl - i*500)
        j    = (rl -maxoffset[i])/dr
        max  = round(j)
        max  = 0

        for k in range(len(dataobs[i])-(n+max)):
            misf += (abs(data[k+n]-dataobs[i][k+n])*abs(data[k+n]-dataobs[i][k+n]))*(w[i][k+n])


    return misf

##############################################################################################
def calc_misf_reg(mver, mhor,zlay,tvimod):
    s=2
    reg=0
    reg2=0
    for r in range(len(mver)-s):
        reg += math.pow(math.log10(mver[r+s])-math.log10(mver[r+s-1]),2)
        if (tvimod==2):
            reg += math.pow(math.log10(mhor[r+s])-math.log10(mhor[r+s-1]),2)

    reg += reg2
    if tvimod==2:
        reg /= 2
    return reg

##############################################################################
def mkcommand(model, mver, mhor, layers, zr, tvi, acur, frequencies ,r0,dr,rl):
    filename = 'model' + str(model) + '.nc'
    #for i, file in enumerate(cmpFiles):
    if (tvi=='off'):
        mhor = mver
    if (len(layers)!= len(mver)-1) :
        print 'Error: Each layer needs a spesified value for the resistivity/conductivity'


    zs= zr - 30
    frex='f'+str(model)

    command = 'elcardinal '
    command += 'ofel=' + filename
    command += ' par=' + 'con'
    command += ' acur='+str(acur)
    command += ' method=digf'
    command += ' comp=ex'
    if (model !=0):
        command += ' frex=' + frex
    command += ' zlay='
    for i in range(len(layers)-1):
        command+= str(layers[i]) + ','
    command += str(layers[len(layers)-1])
    command += ' freq=' + str(frequencies[0])
    for i in range(len(frequencies) - 1):
```

```
        command += ',' + str(frequencies[i+1])
    command += ' tvi=' + tvi
    command += ' mver='
    for i in range(len(mver)-1):
        command +=  str(mver[i]) + ','
    command += str(mver[len(mver) - 1])
    command += ' mhor='
    for i in range(len(mhor)-1):
        command +=  str(mhor[i]) + ','
    command += str(mhor[len(mhor) - 1])
    command += ' zr='+ str(zr)
    command += ' zs='+ str(zs)
    command += ' r0='+ str(r0)
    command += ' dr='+ str(dr)
    command += ' rl='+ str(rl)

    return command

######################################################################
if __name__=="__main__":
    main(sys.argv)
######################################################################
```