NTNU – Trondheim
Norwegian University of
Science and Technology

Parameter estimation in a Markov mesh model by reversible jump MCMC simulation

Johanne Norstein

Master of Science in Physics and Mathematics
Submission date: June 2012
Supervisor: Håkon Tjelmeland, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Preface

This Master's thesis, with course code TMA4905 Statistics, is the final part of my Master's degree in Physics and Mathematics at NTNU. The report is written between February 6th and June 29th.

In order to finish this thesis, I have gotten valuable help. First, I would like to thank my supervisor, Håkon Tjelmeland, for help and support. I would also like to thank Vidar Klungre for proof-reading this document.

Johanne Norstein June 28th 2012

Abstract

We have a model for simulating facies values in a rock. We can use the model to find facies structures in a 2-dimensional area, which we can use to find properties of a rock in a petroleum reservoir. The model is a Markov mesh model, with a conditional probability distribution for the facies values, with a set of parameters. By using a training image with known facies values, we can simulate the parameters in the model, and then simulate facies values for a new area.

In this text, we simulate the parameters by using a Reversible jump Markov chain Monte Carlo algorithm. This lets us simulate not only the values of the parameters, but also which parameters that should be present in the model. We use the Metropolis-Hastings algorithm in the simulations.

We use the model with the simulated parameters to make new images with the Markov mesh model. The images should have similar visual appearance as the training image. We are able to make images with some similar qualities as the training image, even though we are not convinced that the parameter values converge.

Samandrag

Vi har ein modell for å finne fasisverdier i ein stein. Vi kan nytte modellen til å finne fasismønstre i eit todimesjonalt område, og vi kan til dømes bruke han til å finne eigenskaper til ein stein i eit petroleumreservoar. Modellen vår er ein Markovnettmodell, med ei vilkårsbunden sannsynsfordeling for fasisverdiene. Vi har parametrar for sannsynsfordelinga som vi kan simulere ved å nytte eit treningsbilete med kjende fasisverdier.

Vi nyttar ei Markovkjedesimulering med reversible hopp for å simulere parametrane. Dette gjer at vi kan simulere både verdiene til parametrane våre, og kva for parametrar som skal vere ein del av modellen. I simuleringa nyttar vi Metropolis-Hasings-algoritmen.

Ved hjelp av dei simulerte parametrane nyttar vi Markovnettmodellen til å simulere nye bilete. Dei nye bileta vi får bør ha liknande utsjånad som treningsbiletet. Vi klarer å få bilete med liknande struktur som treningsbileta, sjølv om vi ikkje er overbevist om at parameterverdiene konvergerer.

Contents

Symbols

x	Vector with the facies values in our grid
x_i	The facies value of the ith element/node in x
$x_{(<i)}$	The facies values of the elements preceding element i
θ	Vector with parameters
θ_i	The ith element in the θ vector
$\pi(x \mid \theta)$	The conditional probability distribution for the vector x given the parameters θ
$\pi(x_i \mid \theta, x_{(<i)})$	The probability distribution for the value in element i given the parameters θ and the values in the preceding elements of i
G_i	The sequential neighborhood of the ith element in our grid
x_{G_i}	The facies values of the elements in the sequential neighborhood of element i
$\pi(x_i \mid \theta, x_{G_i})$	The probability for the value x_i in the grid given the parameters θ and the sequential neighborhood of x_i
γ_i	The area used to estimate the function values for the function f described in Section 6.1.1
z	A vector with values generated from the functions f, g and h described in Sections 6.1.1, 6.1.2 and 6.1.3 respectively.
$\alpha(\theta \mid \tilde{\theta})$	The acceptance probability in the Metropolis-Hastings algorithm
$\tilde{\theta}$	The proposed new vector to replace θ
$\tilde{\theta_i}$	A new value for θ_i, the ith element of the vector θ
$q(\tilde{\theta} \mid \theta)$	The probability of proposing a change from θ to $\tilde{\theta}$
$q(\theta \mid \tilde{\theta})$	The probability of proposing the reverse move, a change from $\tilde{\theta}$ to θ

1 Introduction

In this text, we create a model for simulating facies values in a rock. In geology, the facies value of a rock can give information about different qualities of the rock, for instance the temperature and pressure in which the rock was created. The goal of the text is to be able to simulate facies values for a rock in the underground. Applications of the results is to find structures in rocks in places that are hard to collect samples from, for example in a petroleum reservoir.

The model we make uses a reversible jump Markov chain Monte Carlo (RJMCMC) algorithm. We want to find parameter values for a generalized linear model that gives the probability for a facies value in a specific position in a 2-dimensional grid. The expression for the generalized linear model we use is from Stien and Kolbjørnsen (2011), and we use the same rules for the formulation of the model. We use only two possible facies values in our model.

We start out with a training image, which has values for the facies in a 2-dimensional area. The training image can be collected from a rock on land with similar qualities as the one we are trying to simulate in the underground. By calculating the likelihood of the training image with the parameters we have, and adjusting the parameters by RJMCMC, we end up with suitable parameters for our model. We can then use this to simulate facies values in a new area.

In this report, we first give a general introduction to Bayesian statistics, Markov mesh models and Markov chain Monte Carlo methods. After this, we give a more detailed description of the model we have used in the simulation, and last we show results from the implementation.

2 Bayesian Modeling

In the last 20 or so years, Bayesian statistics has become a more used and understood field in statistics. We distinguish between Bayesian statistics and frequentistic statistics. The frequentist approach is the first statistical approach that is usually taught at universities and high schools world wide. The difference between the two approaches, lies in the understanding of prior information on a system, and also on how the concept of probability is understood. In the frequentist approach, one uses data or measurements to describe a statistical process. One assumes that the data comes from some probability distribution, and considers the parameters in the distribution constant, but unknown. Examples of frequentist approaches include linear models and regression, and maximum likelihood estimations. With the Bayesian approach, one does not

consider the parameters in the distribution as constant, but rather as stochastic variables. The frequentists are not interested in the probability of achieving other parameter values from observations, because only the values actually observed are interesting. In Bayesian statistics, the probability distribution is calculated from what we call a prior distribution, and then adjusted by the data that we observe.

In the next sections, we describe each element in the Bayesian statistics more thoroughly. The book Gamerman and Lopes (2006) has been used as a reference. We look at the theory behind Bayesian statistics, and how we use it in practice. The expressions of a prior distribution, a likelihood and a posterior distribution and the connection between them are explained thoroughly.

2.1 The likelihood

The likelihood in the Bayesian approach is the same as the likelihood in a frequentist approach. The expression for the likelihood is the same as the expression for the probability, but now we look at the probability as a function of the parameters rather than as a function of the observed values,

$$l(\theta) = \pi(x \mid \theta). \tag{1}$$

Instead of looking at the probability of a value x as the right hand side of the expression indicates, we look at the likelihood of obtaining the x-values we observe when we have the parameters θ. This is the reason why likelihoods are often written as $l(\theta)$, to show that we see the likelihood as a function of the parameter vector θ. In a frequentist approach, we are often interested in the parameter values that result in the maximal value for the likelihood, and maximum likelihood estimation is a central part of the analysis. The maximum likelihood estimation is not as central in Bayesian theory, since we are simulating the parameter values for the probability distribution instead.

2.2 Prior distribution

The prior distribution in a Bayesian model, is a mathematical formulation of some prior knowledge we have of the parameters we want to simulate. The prior distribution is a probability distribution on the parameter values. That is, a distribution that describes the probability for each parameter collection. We use the term parameter collection because we might have more than one parameter to estimate. An important question related to Bayesian inference, is where to

get the prior distribution from. Without looking at the observed values, we make a guess on how the different parameter values are distributed. It might be difficult to formulate the knowledge we have as a probability distribution, and in some cases we need to use a "trivial" probability distribution as the prior. In this case, we try to make a non-informative distribution for θ.

For example, if you have a flipping coin, and you want to determine if the coin is a fair coin or not, prior information on this could come from examining the coin, weighing the coin or measuring the coin. But even if we do notice that the coin has one edge that is i bit thicker, or that the edge is more rounded on one side, how do we formulate this as a distribution? There is no right or wrong answer to this, so the appropriate way to formulate this as a distribution is to make a guess on how this would impact the coin, and apply this to make a change to the trivial, or non-informative, distribution. An example of a prior distribution for the result of the coin flipping, could be

$$\pi(\theta) = \text{Be}(\theta;\ \alpha, \beta), \tag{2}$$

where $\text{Be}(\theta;\ \alpha, \beta)$ represents the beta distribution, with parameters α and β.

Even though there is a fair amount of guessing involved, Bayesians consider this prior information important, and include it in the analysis of a process.

2.3 Posterior distribution

In this section, we describe how the posterior distribution is calculated from Bayes theorem. We use Bayes theorem to combine the information we have prior to an experiment, and the data we get from the experiment. Together this gives the posterior distribution, and we get a probability distribution for the parameter θ.

We have Bayes theorem given by,

$$\pi(\theta \mid x) = \frac{\pi(x \mid \theta)\pi(\theta)}{\pi(x)} \propto \pi(x \mid \theta)\pi(\theta), \tag{3}$$

where $\pi(\theta \mid x)$ is the posterior distribution. The posterior distribution described by Bayes theorem, takes into account both the prior distribution and the likelihood.

2.3.1 Conjugate distributions

In Bayesian statistics, a conjugate family of distributions refers to the case where the prior and the posterior distributions are from the same distribution family.

If the posterior distribution comes from the same family of distributions as the prior distribution, they are conjugate distributions. Using the example from the previous section, we can assume a coin having the beta distribution as a prior distribution, and calculate its posterior distribution. The probability density function for the beta distribution is given by

$$f(\theta;\ \alpha, \beta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{\mathrm{B}(\alpha, \beta)}, \tag{4}$$

where $\mathrm{B}(\alpha, \beta)$ is a normalization constant.

The posterior distribution becomes,

$$\pi(\theta \mid x) \propto \pi(x \mid \theta) \times \mathrm{Be}(\theta;\ \alpha, \beta) = \mathrm{Be}(\theta;\ \alpha_2, \beta_2), \tag{5}$$

where α_2 and β_2 are two different parameters in the beta distribution. From Equation (5) we see that the posterior distribution is also a beta distribution, and thus this is a conjugate distribution.

2.4 Discussion

A prior and a posterior distribution is always relative. After calculating a posterior distribution by Bayesian statistics, we can use this as a prior distribution to calculate a new posterior distribution. In this case, we would need to collect new data in order to have a legal prior distribution. The prior distribution should be possible to obtain without knowing any data, and thus we cannot use the same data as we used to produce this prior distribution.

3 Markov mesh models

Using a Markov mesh model is a simple way to obtain a Markov model in a 2-dimensional mesh. We consider a mesh, or grid, with width m and height n. We can give the elements an ordering by traversing the grid in a lexicographical order, from left to right and from the top to the bottom. For this project, we have an integer value for each element. The symbol x_i refers to the value in the ith element. We have that $i \in \{1, \ldots, m \cdot n\}$.

The idea of a Markov mesh model, is to define a Markov chain on the grid, where the value in each element is only dependent on the value in a subset of the foregoing elements. A *sequential neighborhood* for an element in the grid, is the area that the value in the current element depends on. We say that the

element i has the sequential neighborhood G_i. The values in this sequential neighborhood is a vector which we write as x_{G_i}. The sequential neighborhood can be defined in any way we want, as long as the elements in it are preceding element i in the traversal order. That is,

$$j \in G_i \Rightarrow j < i,$$

but not necessarily that

$$j < i \Rightarrow j \in G_i.$$

In Figure 1 we show an example of a sequential neighborhood.

The Markov mesh has a valid Markov property, as explained in Abend et al. (1965). We have a probability of the value in an element i, given as $\pi(x_i \mid \theta, x_{(<i)})$, where $x_{(<i)}$ is the vector of values of elements that are earlier in the traversal order of the vector x, $\{x_1, x_2, \ldots, x_{i-1}\}$, than element i. In our model, this is a conditional probability. We write the Markov property for the values in our mesh as

$$\pi(x_i \mid \theta, x_1, \ldots x_{i-1}) = \pi(x_i \mid \theta, x_{G_i}). \tag{6}$$

We see that we have created a higher order Markov chain for the 2-dimensional grid, and we call this a Markov mesh. This is a simple way to use Markov chains for simulating the values in the elements in the grid, since we can calculate the values for the whole grid by traversing it once.

4 Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) is an algorithm that can be used to simulate the posterior distribution in a Bayesian model. We define a Markov chain and use Monte Carlo to simulate moves in this Markov chain. A thorough introduction to Markov chains can be found in Ross (2006), and we will not discuss properties of Markov chains in detail in this text.

In a Markov chain model, we have a system with a variable that can be in different *states*. We can move between these states according to rules defining the Markov chain. We call such a move a *step* in the Markov chain. In the following sections, our system is a Bayesian model, and we will define a state as a set of parameters in our model. This means that one specific collection of parameter values represents one state.

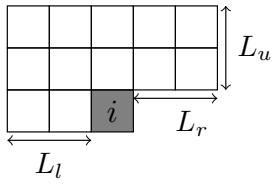


Figure 1: The sequential neighborhood of an element is determined by which elements we want an element to be dependent on. The value in element i is only dependent on the elements in its sequential neighborhood. In this example, the number of elements to the left, L_l, to the right, L_r and above L_u element i we consider, can be chosen independently.

We calculate a posterior distribution from the results we get from the simulation, by starting with a prior distribution that we can obtain by a (qualified or unqualified) guess. The Markov chain part of MCMC is that a step in the process is only dependent on the previous step in the process. The Monte Carlo part is that we choose random numbers from a probability distribution to decide which moves to make in the Markov chain.

4.1 The Markov chain Monte Carlo algorithm

There are various ways to perform an MCMC simulation, and in Section 4.2, we describe one of these, which is called the Metropolis-Hastings algorithm. In this section we will give a more general outline of the algorithm, without many details.

We have a collection of parameters, θ, which has a posterior probability distribution $\pi(\theta \mid x)$. We look at θ as a vector with the parameter values $(\theta_1, \ldots, \theta_M)$, where we have that

$$P(\theta^{n+1} \mid \theta^n, \theta^{n-1}, \ldots, \theta^0) = P(\theta^{n+1} \mid \theta^n), \tag{7}$$

where θ^{n+1} is the collection of parameters after $n+1$ steps in the process. This

is a Markov property, because every iteration in the process is only dependent on the current state of the Metropolis-Hastings algorithm.

4.2 Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm is an algorithm used in the MCMC simulation. We assume that we have a probability distribution, $\pi(\theta \mid x)$ which is hard to sample from. We want to use MCMC, and in particular Metropolis-Hastings to simulate samples from this distribution. Our goal is to simulate the parameters θ, when we know the values for x. Using the problem outline from Section 4.1, we propose a change to the parameter θ_i, for some $i \in \{1, M\}$, in θ, and then we check if we want to accept this change. Let $\tilde{\theta}$ denote the vector of parameters with element θ_i changed to $\tilde{\theta}_i$. The moves between parameter collections define a Markov chain, with limiting probability equal to the distribution of the θs. This way, we simulate a draw from the distribution for θ, and end up with a useful value for the parameters.

We create a Markov chain in the way that we start with some values for the θ parameters, and then we can move one step to a new set of θ parameters. We choose this step to be making a change to one of the parameter values. Assuming we have the θ parameters, we can suggest to instead use a different set of parameters, which we call $\tilde{\theta}$. The proposal distributions, $q(\tilde{\theta} \mid \theta)$ and the reverse $q(\theta \mid \tilde{\theta})$, define the probabilities of proposing a move from the state θ to $\tilde{\theta}$ and opposite, respectively. Multiplying by the probabilities of being in each of these states, we get the *importance ratios* for the states in the Markov chain,

$$\frac{\pi(\theta \mid x)}{q(\theta \mid \tilde{\theta})} \tag{8}$$

and

$$\frac{\pi(\tilde{\theta} \mid x)}{q(\tilde{\theta} \mid \theta)}. \tag{9}$$

We want to find a way to figure out which of the two states we want to continue with, and the Metropolis-Hastings algorithm gives us rules for determining this. The suggestion of moving to the new state $\tilde{\theta}$ is accepted with a probability α, which is given by

$$\alpha(\tilde{\theta} \mid \theta) = min\left(1, \frac{\pi(\tilde{\theta} \mid x)/q(\tilde{\theta} \mid \theta)}{\pi(\theta \mid x)/q(\theta \mid \tilde{\theta})}\right) = min\left(1, \frac{\pi(\tilde{\theta} \mid x)q(\theta \mid \tilde{\theta})}{\pi(\theta \mid x)q(\tilde{\theta} \mid \theta)}\right). \quad (10)$$

In Equation (10), we have that the probability of accepting the change is one if the new state has a higher importance ratio. If not, we get a probability of moving to the new state which is between 0 and 1. In order to decide whether to make the change or not in this case, we draw a number $u \sim \text{Unif}[0, 1]$. The parameter θ is

$$\theta^{n+1} = \begin{cases} \tilde{\theta} & \text{if } u < \alpha(\tilde{\theta} \mid \theta) \\ \theta & \text{otherwise} \end{cases}$$

after iteration $n + 1$.

5 Reversible jump MCMC

Reversible jump MCMC (RJMCMC) is an MCMC variant that does not have a fixed number of parameters. That is, one of the things we need to simulate is the number of parameters in the system. A common way to denote the different steps in a RJMCMC algorithm, is to say that different models are distinguished by having different parameters in them. That is, in RJMCMC, there is a maximum number of parameters that can be a part of a model. We get one model for each possible selection of these parameters to use in the model. This gives us a very large number of models even if there are only a few possible parameters.

The RJMCMC algorithm determines which parameters we want to use, and also gives these parameters a value. Each step in a RJMCMC algorithm contains a proposal part and a selection part. Together, these two result in proposing a new state for the Markov chain we simulate. We then use the Matropolis-Hasings algorithm to decide if we should move to the new state or stay in the state the Markov chain is currently in. Section 5.1 describes how the model proposal is done in RJMCMC, Section 5.2 describes the parameter proposal and in Section 5.3, we put the two together, and discuss how we need to modify the Metropolis-Hastings algorithm for the reversible jump algorithm.

5.1 Model proposal

At the current state in RJMCMC, we have a model. We wish to compare this model with a proposed new model, and in order to do this, we propose to add parameters to or remove parameters from our model. This step in the RJMCMC algorithm can be done in multiple ways. If we for instance have M possible parameters to choose from in a full model (where full model describes a model where all the parameters available are used), we can choose one of these parameters randomly. Now we can check if this parameter is a part of the model in our current state. The proposal model will include the parameter if it is not in the current model. If the parameter is already in the current model, the proposal model can either exclude the parameter, or change its value. This can for instance be done by choosing randomly with a given probability of choosing to remove or change. Notice that the models will have different dimensions if we propose to add or remove parameters.

5.2 Parameter proposal

When we have a proposal model, we can specify the values of the parameters that are in this specific model. If we have just added a parameter, we need to give this parameter a value, and if our proposal model contains the same parameters as the current model, we need to change the parameter that we did not remove. The proposal of the new value can be for instance to draw a normally distributed variable. When we have new values for the parameters, we say that we have a proposed state for the Markov chain.

The proposal probabilities can be calculated easily when we do the proposal in this way, since we assume that choosing the amount of parameters and choosing the value for the parameters are independent.

5.3 State Selection

We decide to accept the proposed state or not by using the Metropolis-Hastings algorithm described in Section 4.2. In some of the cases in RJMCMC, the Metropolis-Hastings algorithm does not only consider different parameters, but also different models. The models are different because of the number of parameters they have, and thus the proposal probabilities $q(\theta \mid \tilde{\theta})$ and $q(\tilde{\theta} \mid \theta)$ will have different dimensions.

The acceptance probability in Equation (10) contains a ratio of proposal probabilities, and probabilities for a given state. We see that the numerator and

the denominator in the main fraction are both dimensionless, since the small fractions have equal dimensions in its numerator and denominator. This means that we can use the Metropolis-Hastings algorithm between different models as well.

6 Our implementation of the reversible jump MCMC algorithm

The implementation of the reversible jump MCMC is an implementation that is based on the article Stien and Kolbjørnsen (2011), but with a reversible jump MCMC implementation. The formula for the probability distribution for the generalized linear model in the article, is also used for the probability distribution, except that we do not consider the parameters in the distribution to be fixed values, but stochastic variables. We simulate the parameter values using RJMCMC rather than estimating them with maximum likelihood estimation.

6.1 Specification of the model

Our model for facies modeling consists of a probability distribution for the facies values in a grid. In this probability distribution, we use values that we get from three functions. Our probability distribution has a collection of parameters, θ, that we want to simulate, and this is done by RJMCMC. The model is a Markov mesh model, like the one explained in Section 3. The value x_i is the facies value in node i, and we have that for all i, $x_i \in \{0, 1, \ldots, K-1\}$. We define a value x_i^k which is given by

$$x_i^k = \begin{cases} 1 & \text{if } x_i = k, \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

for the x_i-s in a training image. We have a training image, with a grid with facies values represented as zeroes and ones. This means that we only have two facies values, so $K = 2$. The training image is used to estimate parameters in a generalized linear model for our probability distribution. We define some functions that use the values in the sequential neighborhood of a node in the training image to give us a probability for the facies value in a specific element. The θ vector is a vector with M elements as well, but in our model, we do not include all of them. The elements not included in the model, are set as *inactive*

parameters, and do not contribute to the probability value. The functions we will describe later relate to the elements in the vector z in the following way:

$$z_i^p = f^p_{x_{\gamma_i}}(x_i), \text{ for } p = \{0, 1, \ldots, P_f - 1\}, \tag{12}$$

$$z_i^p = g^p_{x_{\Gamma_i}}(x_i), \text{ for } p = \{P_f, \ldots, P_f + P_g - 1\}, \tag{13}$$

$$z_i^p = h^p(x_i), \text{ for } p = \{P_f + P_g, \ldots, P_f + P_g + P_h - 1\}, \tag{14}$$

where P_f, P_g and P_h represents the number of function values produced by the functions f, g and h, respectively. The total number of values in the vector z is $P_f + P_g + P_h$ values. The functions f, g and h are defined in Sections 6.1.1, 6.1.2 and 6.1.3, respectively.

The probability for the facies value in the element x_i in our model is given by,

$$\pi(x_i \mid z_i, \theta^1, \ldots, \theta^K) = \frac{\prod_{k=1}^K \exp(z_i^T \theta^k x_i^k)}{\sum_{k=1}^K \exp(z_i^T \theta^k)}, \tag{15}$$

where x_i^k and z_i are as defined above. Here, θ^k describes the values in θ that are associated with the value k. In our case, we have that $K = 2$, and we can write the probability distribution as

$$\pi(x_i \mid z_i, \theta^1, \theta^2) = \frac{\exp((z_i^T \theta^1 x_i^1) + (z_i^T \theta^2 x_i^2))}{\exp(z_i^T \theta^1) + \exp(z_i^T \theta^2)}. \tag{16}$$

The joint probability for the facies values x is

$$\pi(x \mid z, \theta^1, \ldots, \theta^K) = \prod_{i=1}^{m \cdot n} \frac{\prod_{k=1}^K \exp(z_i^T \theta^k x_i^k)}{\sum_{k=1}^K \exp(z_i^T \theta^k)}, \tag{17}$$

where m is the height and n is the width of our grid with facies values.

6.1.1 Combination of the nearest neighbors

The first function we describe considers the values in the four nearest neighbors of the node we are calculating the probability of a value for. In Figure 2, we see the nodes we are interested in. We denote this area γ_i.

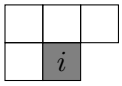


Figure 2: The four nearest elements of the ith element in the mesh. We denote the area γ_i.

We consider all possible combinations of facies values in the four nearest neighbors. Since we know all the foregoing facies values, we know which combination of values is actually present in our grid. All the function values from this function will be zero, except for the value where the combination of nearest neighbors matches the one that is present in the estimated grid. We have the expression for the function,

$$f_{x'_{\gamma_i}}(x_i) = \begin{cases} 1 & \text{if } x'_{\gamma_i} = x_{\gamma_i} \\ 0 & \text{otherwise} \end{cases}, \tag{18}$$

where the x'_{γ_i}s represent different combinations for the values of the elements in the four nearest nodes, γ_i. This function gives us

$$P_f = K^4$$

values in the z_i-vector.

6.1.2 The values in the few nearest nodes

In this function, we consider not only the nearest nodes, but also an area around the nodes that can be larger. The area we consider is shown in Figure 1, and we call his area Γ_i. We can specify values L_l, L_r and L_u, which describe how many nodes to the left of, right of and above our node we want to consider. In this function, we do not consider the combination of the nodes, but we consider each of the nodes by itself. We get one value for each possible facies value for each node in the area Γ_i, and register which value they all have. The function gives a value of 1 if the node we are looking at has the same value as the k value

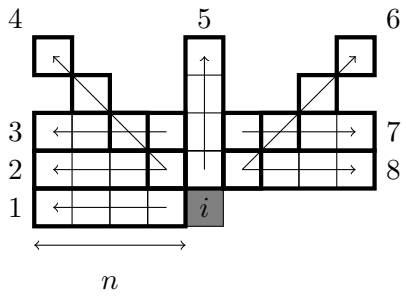


Figure 3: The directions we search in to find succeeding elements of the same value.

in $g^k_{x_j}(x_i)$. We write the function as

$$g^k_{x_j}(x_i) = \begin{cases} 1, & \text{if } x_j = k, \\ 0, & \text{otherwise,} \end{cases} \tag{19}$$

where $x_j \in \Gamma_i$ and $k \in \{0, 1, \ldots, K-1\}$. We get

$$P_g = K \cdot (L_l L_u + L_r L_u + L_l + L_u)$$

values from this function.

6.1.3 Equal values in straight lines in different directions

Here we try to catch the connection between many equal values in each of the directions in Figure 3. We look at interactions in different levels, that is, we consider values that are 2, 3 and up to a maximal depth d places away from the current node. We get one value for each direction in each "depth" of interaction.

We have that

$$h^k_{j,l}(x_i) = \begin{cases} 1 & \text{if all the } l \text{ values in direction } j \text{ are equal and equal to } k, \\ 0 & \text{otherwise,} \end{cases} \tag{20}$$

Table 1: Table illustrating the Reversible jump algorithm. The steps here illustrate the cases described in Section 6.3

Choose one of the parameter values in θ, call it θ_c		
The chosen θ_c value is inactive.	The chosen θ_c value is active	
Case 1	Case 2.1	Case 2.2
Propose to make θ_c active	Propose to make θ_c inactive and with value 0	Propose to keep θ_c active and change the value
Use Metropolis-Hastings to determine which of the parameters to keep		

for all values j in Figure 3 and all interactions l larger than or equal to 2 and up to d. This function gives us

$$P_h = 8K(n-1)$$

values in the z_i-vector.

6.2 The reversible jump MCMC algorithm

We refer to Equation (10) for the expression for the acceptance probability in our MCMC model. In this section, we define the proposal probabilities, $q(\tilde{\theta} \mid \theta)$ that apply to our RJMCMC algorithm.

We have two possible situations for each of the parameters θ_i in the θ-vector. We can say that the parameter is either active or inactive. If the value is active, it has a value, which we can simulate by the MCMC algorithm. If it is inactive, we describe the model as if the parameter is not a part of the model. This means that we switch between two different models by changing a θ parameter from being active to being inactive or opposite. This is the main difference between MCMC and reversible jump MCMC. We have two parts of the reversible jump MCMC algorithm, the step where we try to find the best model, and the step where we calculate the parameters for the model we have chosen. The latter step uses Metropolis-Hastings as described in Section 4. Table 1 shows the different cases we can have in our model.

We consider three different cases for proposing a change from one vector θ to a different one. In our case, this means changing one of the values θ_i in the vector, or adding or removing one such value. The value of the expression for the proposal $q(\tilde{\theta} \mid \theta)$ varies in the different cases. In our algorithm, we draw a

random number c, which is an integer between 0 and M, the maximum number of parameters in θ. We choose the θ value that is in position c in the parameter vector, and we call this θ_c. We determine which case we have by checking if θ_c is active or not. If the value is inactive, we only have one option to propose, and that is to make the parameter active and give it a value. If we choose a θ parameter that is already active, we need to decide whether we want to make it inactive, or keep it active and make a change to the value of the parameter. We have equal probabilities of proposing to make the value inactive and keeping it active.

6.3 The probabilities for proposing a move from θ to $\tilde{\theta}$

In the following sections, we present the proposal probability $q(\tilde{\theta} \mid \theta)$ for different cases. We assume that we have already chosen a θ_c value to consider. We see which of the cases we need to use from this chosen parameter.

Case 1: The θ value we chose is inactive.

The probability of choosing this specific θ_c value is simply one out of the possible number of parameter values and facies values. For this case, we will have M possible parameter values alltogether, since we have M different possible parameter values in θ. Since we also give θ_c a new value (change it), we need to include the probability of choosing this exact value. We denote the proposed changed value as $\tilde{\theta}$. That is, we have that $\tilde{\theta}$ is a new value for θ_c, which we got from choosing a number from a normal distribution with mean zero and variance σ^2. This means that the potential new value for θ_c will be

$$\tilde{\theta_c} \sim N(0, \sigma^2). \tag{21}$$

Alltogether, we get the probability for proposing a move from an inactive value to an active value with value $\tilde{\theta_c}$,

$$q(\tilde{\theta} \mid \theta) = \frac{1}{M} \cdot \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(\tilde{\theta_c})^2}{2\sigma^2}\right). \tag{22}$$

Case 2.1: The θ value we chose is active, and we propose to make it inactive.

If the θ value we choose is active, we can either propose to make it inactive, or propose to change it. In our algorithm, we simply choose between these two by drawing with equal probabilities for both outcomes. Specifically, we draw a uniformly distributed random number, where a value below 0.5 results in proposing to make the parameter value inactive, and a number above 0.5 results in proposing to keep the parameter value active, and give it a new value.

We have the following probability of proposing to make a value inactive,

$$q(\tilde{\theta} \mid \theta) = \frac{1}{2M}. \tag{23}$$

Case 2.2: The θ value we chose is active, and we propose to keep the value active, and make a change to the value.

If we choose an active value, and propose to keep it active, we need to make a change to the value. We propose to make a change to the θ_c value, by drawing a value from the normal distribution with mean θ_c and variance σ^2,

$$\tilde{\theta_c} \sim N(\theta_c, \sigma^2). \tag{24}$$

We thereby get the proposal probability

$$q(\tilde{\theta} \mid \theta) = \frac{1}{2M} \cdot \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(\theta_c - \tilde{\theta_c})^2}{2\sigma^2}\right). \tag{25}$$

6.4 The probabilities for proposing the reverse move, a move from $\tilde{\theta}$ to θ

In the Metropolis-Hastings algorithm we compare the current parameter value with the proposed new value. We use the probability of proposing the changes between the parameter values to compare the values, and therefore we need to consider the probability of making the move from $\tilde{\theta}$ to θ as well.

Case 1: The θ value is inactive, and $\tilde{\theta}$ is active.

We want to look at the probability of choosing a specific θ value. To consider the probability of proposing the opposite move of θ to $\tilde{\theta}$, we assume that we are in $\tilde{\theta}$, and find the probability of proposing to go back to θ. That is one out of the number of possible θ values, or the number of possible parameters we have (M). Since we have a probability of 0.5 for keeping the parameter value active instead of making it inactive, we need to account for this in the expression. The probability of proposing this θ_c and make it inactive is,

$$q(\theta \mid \tilde{\theta}) = \frac{1}{2M}. \tag{26}$$

Case 2.1: The θ value is active, and $\tilde{\theta}$ is inactive.

If our proposed move is to make the θ_c value we choose inactive, we need to look at the probability of proposing the reverse move. We assume that we have the inactive parameter $\tilde{\theta_c}$, and want to find the probability of proposing to make this parameter active value with value equal to the value of θ_c. The probability of proposing this exact value for this exact parameter, is given as,

$$q(\theta \mid \tilde{\theta}) = \frac{1}{2M} \cdot \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-\theta_c^2}{2\sigma^2}\right). \tag{27}$$

Case 2.2: The θ value we chose is active, and $\tilde{\theta}$ is active, but different.

For the case where we have proposed to keep the parameter value active and give it a new value, we need to look at the probability of proposing to go back to the value we had originally. We assume that we have the value for $\tilde{\theta_c}$ and propose to move to θ_c. The probability of doing this is,

$$q(\theta \mid \tilde{\theta}) = \frac{1}{2M} \cdot \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(\tilde{\theta_c} - \theta_c)^2}{2\sigma^2}\right). \tag{28}$$

We see that this expression will get the same value as the expression for $q(\tilde{\theta} \mid \theta)$ for this case. This means that in the expression for α, these values will cancel.

6.5 Our prior distribution

The posterior distribution in Equation (17), is composed by the probability we have for $\pi(x \mid \theta)$ and our prior distribution $\pi(\theta)$ in the following way,

$$\pi(\theta \mid z, x) \propto \pi(\theta)\pi(x \mid z, \theta), \tag{29}$$

where we will define $\pi(\theta)$ later in this section.

The prior distribution we have chosen in our model, divides the θ values into groups, depending on which of the functions f, g and h it corresponds to. In Equation (17) we see that the θ vector is multiplied by the z vector, and each of the elements here correspond to one of the functions. Since our model is based on that the value in a node is dependent on the nearest nodes, we want to give the θ parameters corresponding to function f a higher probability of being active. We say that these parameters are in group 1. The θ parameters corresponding to functions g and h are in group 2. The prior probabilities of the different groups are,

$$P_i(\theta_i \text{ is active}) = \begin{cases} 0.9 & \text{if } \theta_i \text{ is in group 1,} \\ 0.5 & \text{if } \theta_i \text{ is in group 1,} \end{cases} \tag{30}$$

where we have chosen to give the parameters in group 2 a probability of being active of 0.5 because this indicates that we do not know very well if they should be in the model or not.

When we have an active value, we a priori assign this parameter a value that is normally distributed with mean zero, and a variance σ_0^2. We write this as

$$\theta \mid (\theta \text{ is active}) \sim N(0, \sigma_0^2), \tag{31}$$

where we have that σ_0^2 is a rather large variance.

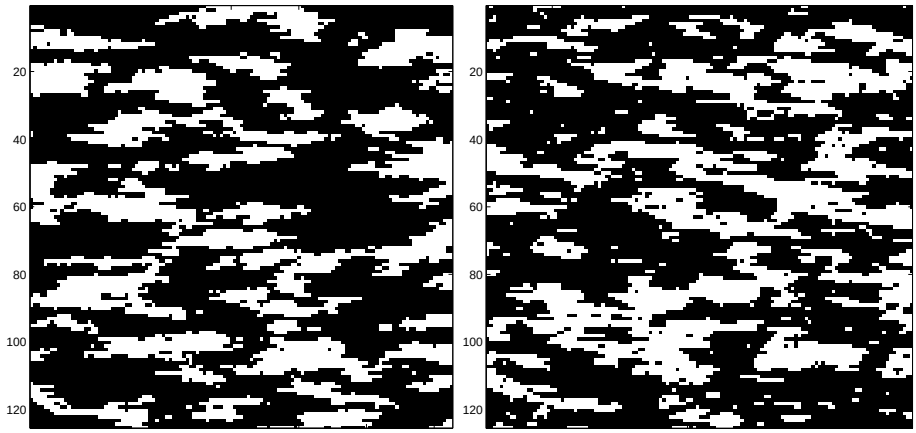
Equations (30) and (31) give us the following prior probability for a θ value,

$$\pi(\theta_i) = \begin{cases} P_i \cdot \frac{1}{\sqrt{2\pi}} \frac{1}{\sigma_0} \exp\left(-\frac{\theta_i^2}{2\sigma_0^2}\right) & \text{if } \theta_i \text{ is active,} \\ 1 - P_i & \text{if } \theta_i \text{ is inactive.} \end{cases} \tag{32}$$

The θ values are considered independent, and thus we get the prior distribution,

$$\pi(\theta) = \pi(\theta_1)\pi(\theta_2)\cdots\pi(\theta_M), \tag{33}$$

with the probabilities for each θ_i given in Equation (32).



(a) (b)

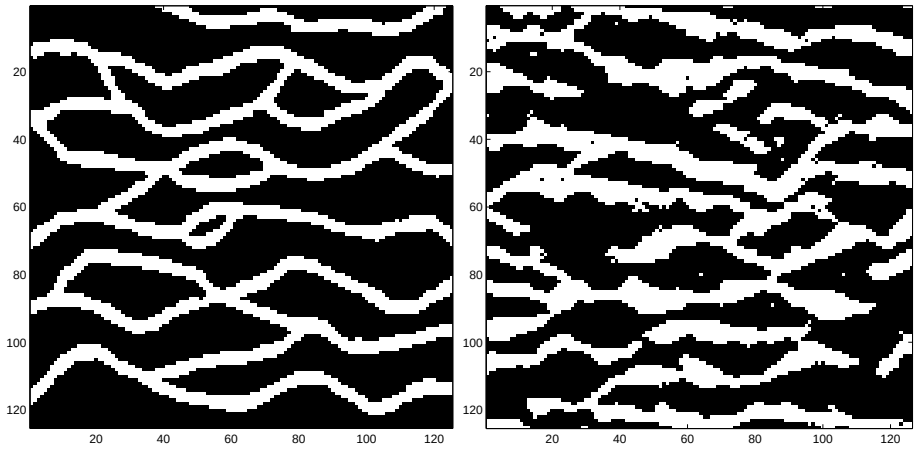
Figure 4: The Sisim training image (a) and the resulting image from the simulation of parameters based on the Sisim image (b).

7 Results from the simulations

In this section, we present the results we get from the Reversible jump MCMC simulations and the evaluation of the model with simulated parameters. We simulate parameters for three different training images. The first one, which we from now on call the Sisim training image, has facies values that are concentrated in certain areas. We see this training image in Figure 4 (a). The second training image has a channel like structure, and is shown in Figure 5 (a). We call this the Channel training image. The third one, has an ellipsoid like structure, and we call it the Ellipsoid training image. Figure 6 (a) shows this training image.

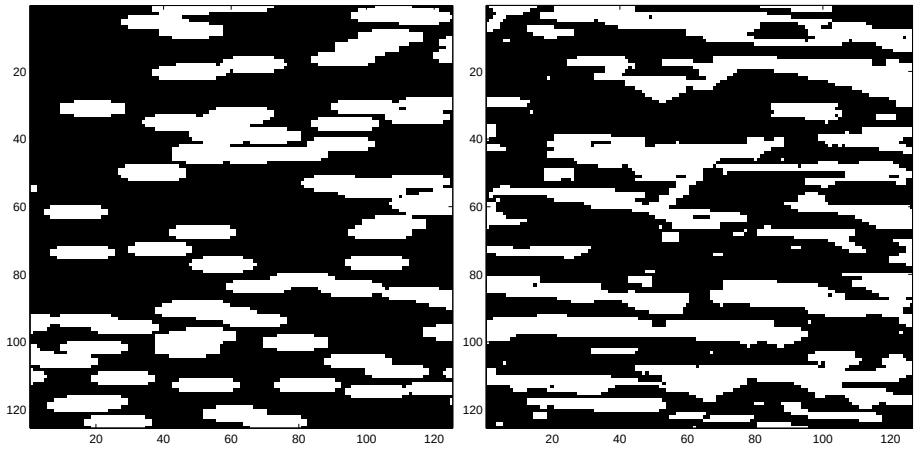
The goal of simulating the θ vector is to get parameter values for the Markov mesh model with probability distribution given in Equation (15). We simulate a realization of the probability distribution $\pi(x \mid \theta)$ with our parameter vector, which results in a simulated image. This resulting image is expected to have a similar visual appearance as the training image used to simulate the parameter for this θ vector.

In order to avoid boundary effects, we use a large grid in the simulation of the facies values. It is hard to determine exactly how large this grid needs to



(a) (b)

Figure 5: The Channel training image (a) and the Channel resulting image (b).



(a) (b)

Figure 6: The Ellipsoid training image (a) and the Ellipsoid resulting image (b).

Table 2: Table showing some of the values of the different input parameters for the algorithm that were tested before the simulations. The variance, σ^2, is set to 1 on all the runs, since we already found this value to be reasonable from previous runs which are not included in the report.

	σ	L_l	L_r	L_u	d	Grid size
Run 11	1	2	2	2	3	2
Run 12	1	3	3	3	3	2
Run 13	1	3	3	2	4	2
Run 14	1	2	2	2	4	2
Run 15	1	4	4	4	4	2

be, but we know that we at least need the size of the sequential neighborhood in order to get good values for the z values in our probability distribution. In this project, we use a grid that is twice as large as the training image. We look at the bottom middle area of this grid, and use the structure here to compare with the training image. In a real world problem, we would want to use a grid that covers the area that we want to simulate facies values for, and then expand it with an area significantly larger than the sequential neighborhood in each direction. In the next section, we will show the results from the parameter simulations and the images we evaluate from these.

Before we perform the simulations, we do trial simulations with fewer iterations in order to optimize some of the input parameters for the algorithm. We have some inputs that affect the simulations, for instance the variance, σ^2, in the normal distribution we draw from when we propose a change to one of the parameters in the model. Other inputs that can vary, are the areas that we use the functions g and h on, and the size of the grid that we evaluate after the parameters are simulated. These simulations are all performed with the Sisim image as training image. We use the inputs from the test simulation that gives the best result on all the final simulations. This means that we do not optimize the inputs for each of the training images individually, but use the ones we get from the Sisim training image. One reason for this is to test if the model needs a lot of adjustment for different training images. Some of the input parameters that were tested are shown in Table 2.

The resulting images from the runs with the input parameters given in Table 2 are shown in Figure 7. The runs were performed with 100 000 iterations, which should be enough to give reasonable results. We will consider convergence issues

Table 3: The real runs, with the parameters used. All runs have been done with 500 000 iterations.

Training image	σ	L_l	L_r	L_u	d	Grid size
Sisim image	1	3	3	3	3	2
Channel image	1	3	3	3	3	2
Ellipse image	1	3	3	3	3	2

when we do the final simulations. We have also included the training image in the figure.

The images do not have a very clear candidate for which of them that captures the structure the best. We choose to continue with the image that has a medium number of possible parameters, which allows us to have a reasonable running time, but still many choices of parameters. The middle left image captures the structure reasonably well, and we choose to use the inputs from this run, that is Run 12 in Table 2, for the remaining runs.

The final simulations are performed with the parameters that are given in Table 3. We present the results for the different training images in the next section. All of the runs are performed with 500 000 iterations.

7.1 The Sisim training image

The Sisim training image has been discussed briefly in the previous section. The result for this training image is shown in Figure 4 (b). For the final run, we see that a fair amount of the structure has been captured. This is also expected, since this is the image we have optimized the input parameters for. One issue that is not captured by the results is that there should not be any single points in the image. We assume that there are some dependencies our model is not able to detect that is the reason for these inaccuracies.

7.2 The Channel training image

In this training image, we want to capture the structure which is horizontally continuous, and is connected. We see in Figure 5 (a) that the structure is mainly connected in the resulting image, which means that some of the structure is captured. The connected channels in the resulting image in Figure 5 (b) are quite thick compared with the training image, and this can indicate that the param-

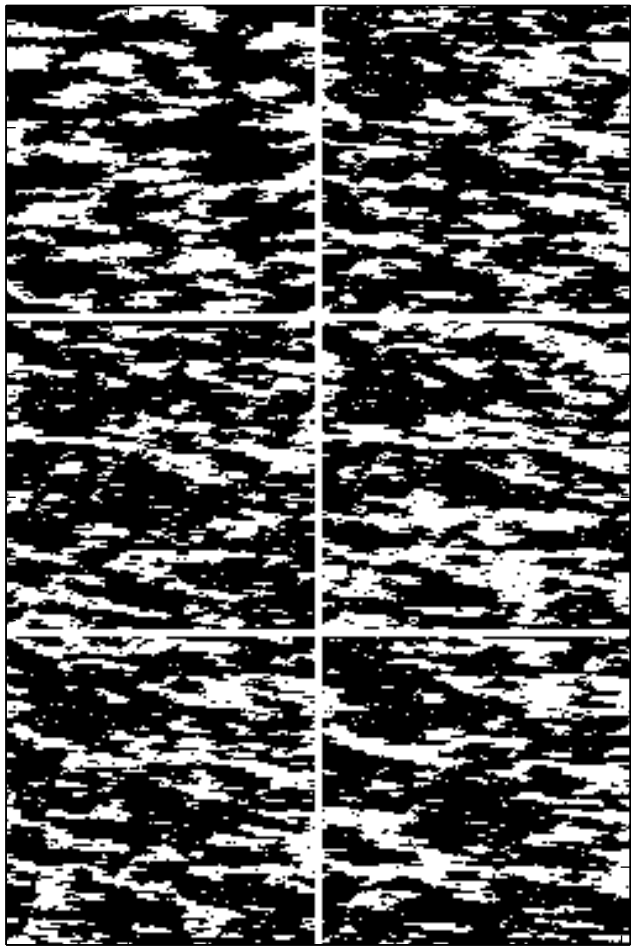


Figure 7: The resulting images for the runs with parameters given in Table 2. The images are placed such that the top left image is the training image, and the top right image is the image with parameters from Run 11 in the table. We want to use the parameters from the resulting image that has the most similar structure as the training image.

eters from function h, which consider if we have the same facies values in one specific direction, may have been too heavily weighted. Compared to the results in Stien and Kolbjørnsen (2011), the results are quite good. This is a good sign, since we used the model from there, and thus some of the imperfections may be caused by the model, and not necessarily the parameter simulation.

7.3 The Ellipsoid training image

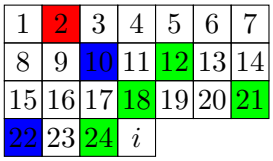
For this training image, we want to capture ellipsoid shaped structures. The results here are not very good, as we can see in Figure 6 (b). The structures captured are connected in many areas where there should have been some space between the ellipsoid shapes. Some round structures are captured, which shows that the algorithm has been able to detect some of the properties. In the next section we discuss which of the parameter values that are active, and in Figure 9 we can see that the Ellipsoid training image is distinguished from the other training images in that fewer of the parameters connected with the h function are active.

7.4 The active θ parameters and convergence

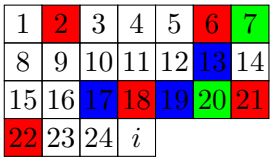
It is interesting to look at which of the parameters that were chosen to be active in the simulations. One of the goals of using the Reversible jump algorithm, is to let the algorithm choose which of the parameters to use, rather than changing the program for each training image prior to the simulation. After a number of iterations in our RJMCMC algorithm, we stop the iterations and evaluate the model with the parameters that are active at that point.

In Figures 8 and 9, we see which of the nodes in the Markov mesh that correspond to the parameters that are active after a run of our algorithm. Figure 8 illustrates which of the nodes for the function g described in Section 6.1.2 that are active. We do not see an obvious pattern for the parameters that are active, but we do notice that there are active parameters even in the boundaries of the area we have used, which means that we would have gotten a different set of parameters by using a smaller area for the area Γ_i.

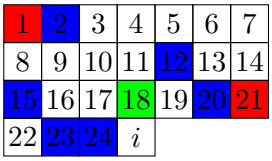
The parameters that were active for the function h described in Section 6.1.3, are shown in Figure 9. Recall that the function h tells us if the nodes in a specific direction have equal values or not. For our runs, we have used a depth of 3, which means that we get values for the 2- and 3-node interactions in each direction. The values in nodes numbered 9-16 in Figure 9 represent that it is significant if all the values in this direction up to the node have the same value



(a)

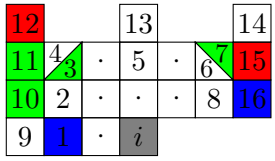


(b)

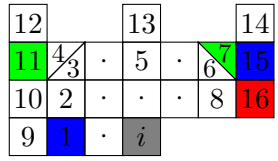


(c)

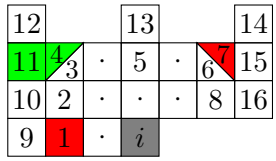
Figure 8: The nodes corresponding with the active θ values for the nodes considered in function g. For the Sisim (a), Channel (b) and the Ellipse (c) training image. The green nodes are active for the $k = 0$ facies value, the blue for the $k = 1$ facies value, and the red nodes are active for both facies values.



(a)



(b)



(c)

Figure 9: The nodes corresponding with the active θ values for the nodes considered in function h. For the Sisim (a), Channel (b) and the Ellipse (c) training image. The green nodes are active for the $k = 0$ facies value, the blue for the $k = 1$ facies value, and the red nodes are active for both facies values. The colored nodes mean that the parameter that corresponds to whether the values up to this node (with the directions shown in Figure 3) have the same facies value. Here, direction 3 corresponds to the numbers 3 and 11, and direction 4 corresponds to the numbers 4 and 12. Similarly with the numbers 6 and 7.

or not. Most of the active parameters consider 3-node interactions. This is not very surprising, as we would think that it is more important to know if three nodes have the same values than if only two nodes have the same value.

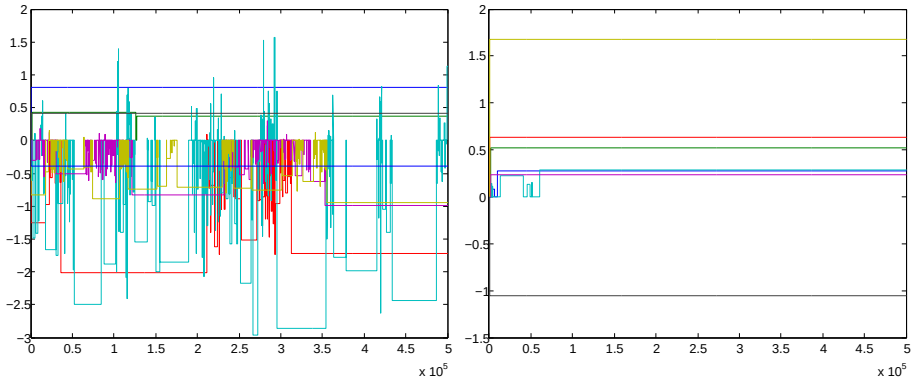
We look at the parameters that are active for more than 70 percent of the time, after what we call burn-in time. The burn-in time is the time it takes for most of the values to stabilize, or at least have been through some propose and accept/reject steps. The values that are active when we stop the iterations are exactly the same as these values. All of the parameters that are active when we stop the iterations are in fact active for more than 90 percent of the time. Many of the parameters have never been inactive, which might be a concern, because in a Reversible jump algorithm, we do want there to be small probability that a value is made inactive even if the value it has is very probable. If this had been the case with only one parameter, we would not be concerned, but it is unlikely that this should happen with many of the parameters.

We do not only need the active parameters to converge, but also the values of these parameters. A plot of the parameter values that are active after the simulations for the Sisim training image are given in Figure 10. The plots for the other runs are very similar. Many of the parameter values, especially in Figures 10 (b) and 10 (c) seem to never change. This illustrates the concerns mentioned in the previous paragraph, the algorithm seems to have been stuck in these values. The plots indicate that the parameter values might not have converged, which might explain some of the imperfections of the resulting images.

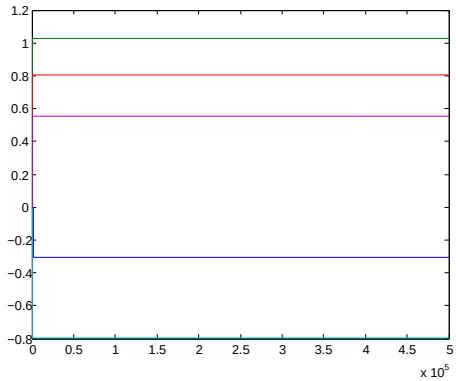
8 Closing remarks

The implementation of the Reversible jump Markov chain Monte Carlo algorithm is able to capture some of the structure from the training images. The Reversible jump algorithm is able to choose only a few of the parameters to be active. This results in a model with fewer parameters, which is good from a statistical point of view. Even though we cannot be sure the parameter values have converged, the results are satisfying to some degree. Compared with the maximum likelihood model in Stien and Kolbjørnsen (2011), the results are approximately equally good, based on the structure that is captured on resulting images.

An advantage of the Reversible jump MCMC algorithm is that we do not need to do human work in order to use it on different training images, and that we end up with a smaller number of parameters. The computation time of the RJMCMC algorithm is rather large, and we need more computation time



(a) (b)



(c)

Figure 10: The values of the active theta parameters divided into the values connected with function f (a), function g (b) and function h (c) for clarity.

in order to simulate the parameters. The program that was created for this project could be optimized, and thus reduce the computation time. However, the computation time will always be larger than for a maximum likelihood estimation.

An idea to future work on this subject is to optimize the code to run faster. We could also have extended the model to 3 dimensions or increased the number of facies values we want to consider, and thus get a more complicated model. This would of course require new training images. We could also have tried to find an algorithm with better convergence properties.

References

Abend, K., Harley, T. and Kanal, L. (1965). Classification of binary random patterns, *IEEE Transactions on Information Theory* **11**(4): 538–544.

Gamerman, D. and Lopes, H. (2006). *Markov chain Monte Carlo: stochastic simulation for Bayesian inference*, Texts in statistical science, Taylor & Francis.

Ross, S. M. (2006). *Introduction to probability models, Ninth Edition*, Academic Press, Inc., Orlando, FL, USA.

Stien, M. and Kolbjørnsen, O. (2011). Facies modeling using a Markov mesh model specification, *Mathematical Geosciences* **43**(6): 611–624.