DTU Library

An Implementation of the Frequency Matching Method

Lange, Katrine; Frydendall, Jan; Hansen, Thomas Mejer; Zunino, Andrea; Mosegaard, Klaus

Publication date:
2013

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Lange, K., Frydendall, J., Hansen, T. M., Zunino, A., & Mosegaard, K. (2013). *An Implementation of the Frequency Matching Method*. Technical University of Denmark. DTU Compute Technical Report-2013 No. 09

An Implementation of the Frequency Matching Method

Katrine Lange*, Jan Frydendall, Thomas Mejer Hansen,
Andrea Zunino, Klaus Mosegaard

*DTU Compute, Technical University of Denmark,
Matematiktorvet, Building 303B, 2800 Kongens Lyngby, Denmark
Center of Energy Resources Engineering, Technical University of Denmark,
Søltofts Plads, 2800 Kongens Lyngby, Denmark*

Abstract

During the last decade multiple-point statistics has become increasingly popular as a tool for incorporating complex prior information when solving inverse problems in geosciences. A variety of methods have been proposed but often the implementation of these is not straightforward. One of these methods is the recently proposed Frequency Matching method to compute the maximum a posteriori model of an inverse problem where multiple-point statistics, learned from a training image, is used to formulate a closed form expression for an a priori probability density function.

This paper discusses aspects of the implementation of the Frequency Matching method and the techniques adopted to make it computationally feasible also for large-scale inverse problems. The source code is publicly available at GitHub and this paper also provides an example of how to apply the Frequency Matching method to a linear inverse problem.

Keywords: multiple-points statistics, training image, a priori information, maximum a posteriori model

---

*Corresponding author, email address: katla@dtu.dk

1 Introduction to the Frequency Matching Method

The Frequency Matching (FM) method defines the maximum a posteriori model of an inverse problem using multiple-point statistics learned from a training image (TI) as prior information. Inverse problems having such type of a priori information arise in scientific fields involving modelling of unknown parameters describing spatial properties. They are typical in the geosciences where, for instance, a property of the subsurface of the Earth should be modelled. The available data is often scarce, resulting in severely under-determined inverse problem. This makes use of a priori information even more beneficiary.

A priori information is often available as expectations to the subsurface showing certain structures, and when modelling spatial properties these structures are provided by so-called training images. Models of parameters describing spatial properties are often referred to as images, letting the colours of the image illustrate the values of the property they are describing.

The Frequency Matching method was introduced by Lange et al. (2012), to which we refer for a detailed motivation for the method and discussion of the general use of multiple-point statistics when solving inverse problems in the geosciences. The present paper describes a Fortran implementation of the FM method with the purpose of making other users able to use the Fortran version of the FM method on their respective problems.

1.1 Probabilistic Inverse Problem Theory

The FM method defines the maximum a posteriori model for an inverse problem, i.e., the model with maximum a posteriori probability, using multiple-points statistics learned from a training image as a priori information. To do so it formulates a closed form expression of the a priori probability density function, which is based on a distance measure, $c(\mathbf{m}, \mathbf{m}^{TI})$, from a model or image $\mathbf{m}$ to a training image $\mathbf{m}^{TI}$. The distance measure expresses how dissimilar the multiple-point statistics of the images are. Models with multiple-point statistics similar to the multiple-point statistics of the training image have short distances to the training image and they are therefore assigned high probabilities. Likewise models with dissimilar multiple-point statistics will have large distances to the training image and they will there-

fore be assigned low probabilities.

The inverse problem is formulated using probabilistic inverse theory (Tarantola, 2005). The data misfit gives rise to the likelihood of a model. Assuming Gaussian measurement noise of the data observations $\mathbf{d}^{\text{obs}}$ with mean zero and covariance matrix $\mathbf{C_d}$, the likelihood function L is defined as:

$$L(\mathbf{m}) \;=\; \text{const} \exp\left(-\frac{1}{2}\|\mathbf{d}^{\text{obs}} - g(\mathbf{m})\|_{\mathbf{C_d}}^2\right),$$

where g is the mapping operator from model space to data space and const is a constant.

Using the distance function c the FM method defines a closed form expression for the a priori probability density function ρ of a model as:

$$\rho(\mathbf{m}) \;=\; \text{const} \exp\left(-\alpha\, c(\mathbf{m}, \mathbf{m}^{TI})\right). \tag{1}$$

According to probabilistic inverse problem theory the posterior probability density function σ is proportional to the product of the prior probability density function and the likelihood function:

$$\sigma(\mathbf{m}) \;=\; \text{const}\; \rho(\mathbf{m})\; L(\mathbf{m}). \tag{2}$$

Specifically using the a priori probability density function from equation (1), the FM method then defines the solution to the inverse problem, $\mathbf{m}^{\text{FM}}$, as the model maximizing the a posteriori probability density function from equation (2)

$$\begin{aligned}
\mathbf{m}^{\text{FM}} &= \operatorname*{argmax}_{\mathbf{m}} \{\sigma(\mathbf{m})\} \\
&= \operatorname*{argmin}_{\mathbf{m}} \{-\log \sigma(\mathbf{m})\} \\
&= \operatorname*{argmin}_{\mathbf{m}} \left\{\frac{1}{2}\|\mathbf{d}^{\text{obs}} - g(\mathbf{m})\|_{\mathbf{C_d}}^2 + \alpha\, c(\mathbf{m}, \mathbf{m}^{TI})\right\}.
\end{aligned} \tag{3}$$

The maximisation and minimisation is done over the set of all valid models $\mathbf{m}$. In this set each model parameter (i.e., each element of the model vector $\mathbf{m}$) belongs to a predefined problem specific set of discrete values taking into account hard data constraints. This means the frequency matching model $\mathbf{m}^{\text{FM}}$ is the solution to a combinatorial optimization problem. The FM method does not dictate how this optimization problem should be solved.

1.2 Formulating A Priori Information

The multiple-point statistics of an image is represented by what the Frequency Matching method defines as the frequency distribution. This is a histogram of the counts of the different patterns found in the image. The patterns, if their size is chosen wisely, are assumed to describe the multiple-point statistics of the image that we seek to reproduce.

Assume the multiple-point statistics extracted from the training image can be expressed as patterns of identical geometric shape consisting of $n+1$ voxels. This implies that a voxel is assumed statistically independent of all voxels except the surrounding n voxels together with which the voxel forms a pattern. The entire training image is scanned for patterns and the count of appearances for each pattern is collected. These counts constitute the frequency distribution. Let $\boldsymbol{\pi}^{\mathrm{TI}}$ and $\boldsymbol{\pi}$ be the frequency distributions of the training image and a model image, respectively.

This section briefly defines the dissimilarity function c used in the closed form expression of the a priori probability density function from equation (1). The current choice of dissimilarity function has roots in the statistical literature regarding the chi-square test for homogeneity (Sheskin, 2004).

Let m be the number of categories of voxels in the image then there exist m^{n+1} different patterns. A majority of these will have the count of 0 as they do not appear in the image. Given the frequency distributions of an image, $\boldsymbol{\pi}$, and of a training image, $\boldsymbol{\pi}^{\mathrm{TI}}$ the dissimilarity function value of the image is defined as follows:

$$c(\mathbf{m}, \mathbf{m}^{TI}) = \chi^2(\boldsymbol{\pi}, \boldsymbol{\pi}^{\mathrm{TI}}) \;=\; \sum_{i \in I} \frac{(\pi_i^{\mathrm{TI}} - \epsilon_i^{\mathrm{TI}})^2}{\epsilon_i^{\mathrm{TI}}} + \sum_{i \in I} \frac{(\pi_i - \epsilon_i)^2}{\epsilon_i}, \qquad (4)$$

where the set I consists of indices of patterns that occur in either the image or the training image. ϵ_i denotes the count of the underlying distribution of patterns with the ith pattern value for images of the same size as the image, and ϵ_i^{TI} denotes the counts of the underlying distribution of patterns with the ith pattern value for images of the same size as the training image. These counts are computed as:

$$\epsilon_i \;=\; \frac{\pi_i + \pi_i^{\mathrm{TI}}}{n^{\mathcal{Z}} + n^{\mathrm{TI}}}\, n^{\mathcal{Z}},$$

$$\epsilon_i^{\mathrm{TI}} \;=\; \frac{\pi_i + \pi_i^{\mathrm{TI}}}{n^{\mathcal{Z}} + n^{\mathrm{TI}}}\, n^{\mathrm{TI}},$$

4

where $n^{\mathcal{Z}}$ and n^{TI} are the total number of counts of patterns in the frequency distributions of the image and the training image.

2 The Implementation

2.1 Assumptions of the Implementation

The FM method itself has no limitations regarding non-linearity but the current implementation assumes that the inverse problem is linear:

$$\mathbf{Gm} = \mathbf{d}^{\mathrm{obs}}, \tag{5}$$

Here $\mathbf{G}$ is a known system matrix, $\mathbf{d}^{\mathrm{obs}}$ is a set of observed data values and $\mathbf{m}$ is the model parameters to be determined.

The FM assumes that these model parameters can take only a limited number of categorical values. Often the model parameters are binary. This can for instance be the case when modelling the flow of the subsurface. The model parameters are then either 0, which represents zones with high permeability and therefore easy flow, or 1, which represents low-permeable zones. Let $\mathsf{sV} + 1$ be the number of categories voxel values can belong to, i.e., for a binary image $\mathsf{sV} = 1$. Per definition the voxel values of the images are $0, \ldots, \mathsf{sV}$.

The FM model is defined as the minimiser of the negative logarithm of the posterior probability density function as given by (3). The minimization is in the current implementation performed using simulated annealing. We will not go into details about the choice of this optimization method or the method itself, but for more information on simulated annealing see Kirkpatrick et al. (1983). Pseudo code for applying simulated annealing to the FM is available in Lange et al. (2012).

2.2 Overview of the Procedures

Figure 1 shows an overview of the most important interactions among the Fortran procedures in the implementation of the FM method. For now we will provide a short walk-through of the procedures. A description of what each of them do is provided in A.

The implementation is based on the FMM procedure which primary function is to set up and initialize all the inputs for the FM method and the
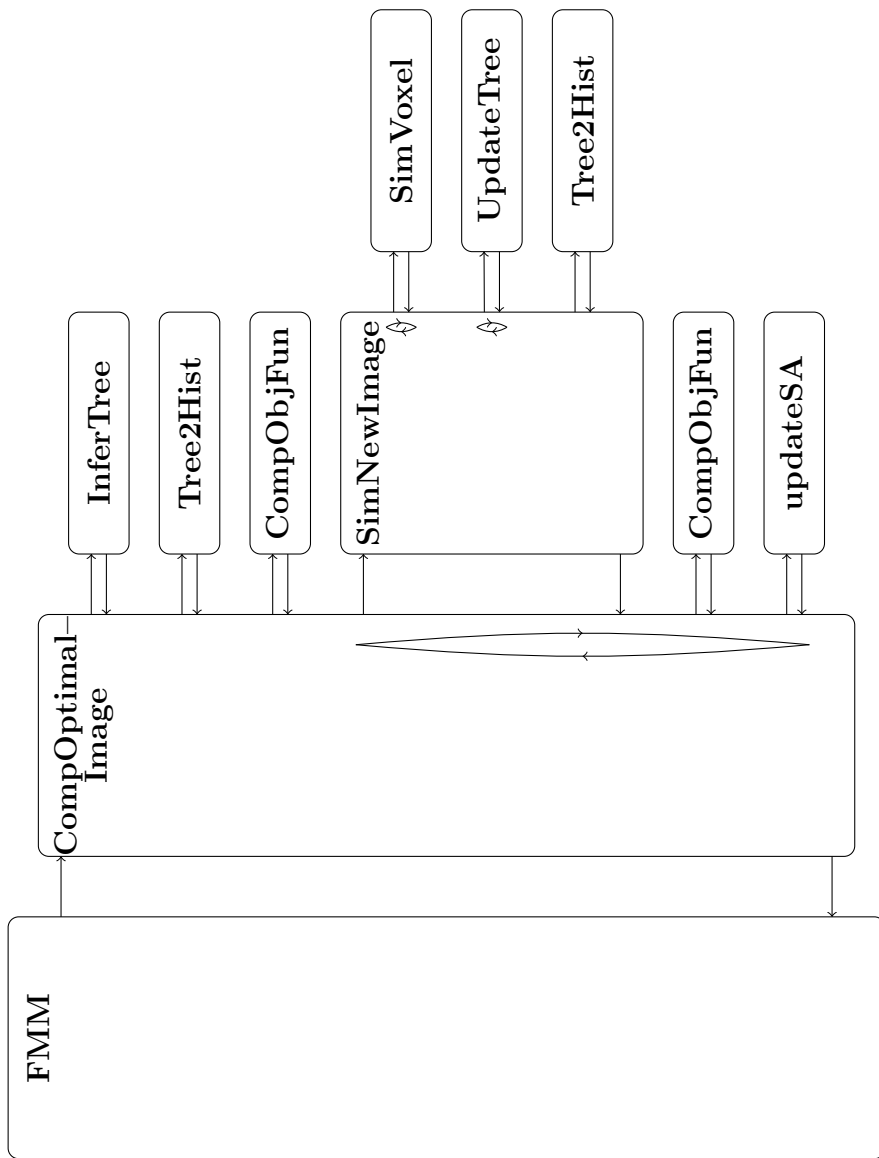
Figure 1: Overall structure of the FM implementation.

simulated annealing scheme. The FMM procedure also has the task of extracting multiple-point statistics of the training image and representing it using the—for that purpose designed—tree structure. (The tree structure will be explained in details in section .) The procedure also generates the frequency distribution. It is the only procedure that uses the training image itself, as it passes on only the tree and its frequency distribution.

The simulated annealing is implemented in the CompOptimalImage procedure. Provided with an initial model, the first thing the CompOptimalImage procedure does is to extract the multiple-point statistics of the initial model (InferTree) and compute its frequency distribution (Tree2Hist). It then computes the objective function value of the initial model (CompObjFun). The CompObjFun procedure calls two other procedures, CompChiDist and CompDataFit, to compute each of the two terms of the objective function.

Vertical arrows in Figure 1 represent a loop, and in the CompOptimalImage procedure it is used to loop through the iterations of the simulated annealing algorithm.

For each iteration a perturbed image is generated by SimNewImage. This is done by erasing the voxel values in a part of the image and then re-simulating them using sequential simulation (Guardiano and Srivastava, 1993) with the multiple-point statistics learned from the training image (SimVoxel). Each time a voxel has been re-simulated the tree must be updated to fit the new image (UpdateTree). Here again the vertical arrows represents loops indicating that these two tasks are done voxel by voxel.

Afterwards, the frequency distribution of the perturbed image is computed (again Tree2Hist). The objective function value of the perturbed image is then computed (again CompObjFun) and finally, the perturbed model is possibly accepted and the variables updated accordingly (UpdateSA).

The procedures in the implementation that have been left out of the diagram are auxiliary procedures mostly related to operations on trees. The auxiliary procedures are described in B .

2.3   Compiling the Program

The program is written following the Fortran 2008 standard, hence a compiler complying with this standard is needed to run the software. We have used the open source GFortran-version 4.6-compiler from GNU to compile and run the code. To simplify the compilation and linking process a Makefile to be used with the GNU make program is provided.

It has been tested successfully on Mac (10.6 and 10.8) as well as GNU/Linux.

2.4 The Inverse Problem

The FMM procedure takes among other inputs the parameters from Eq.(5). For that we have defined a structure called inverseproblem. It contains the following parameters:

G: 2D array with the system matrix $\mathbf{G}$.

dobs: 1D array with the vector of data observations $\mathbf{d}^{\text{obs}}$.

invCov: 2D array with the inverse of the covariance matrix, $\mathbf{C_d^{-1}}$.

cat: 1D array with sV+1 elements, one for each category of voxel values. This is used to transform the images with categorical voxel values to models with physical parameter values. The model parameter associated with a voxel with value i will be assigned the value $\mathsf{cat(i+1)}$ in order to compute the data fit.

The parameters specifying the inverse problem are passed between procedures in the variable InvProb. InvProb can be used to specify any arbitrary linear problem. The implementation can therefore be applied to inverse problems also in other scientific fields outside of the geosciences.

2.5 Specifying Neighbourhoods

We distinguish between two types of voxels in an image: inner voxels and non-inner voxels. Inner voxels are those that are sufficiently far from the boundaries to have enough neighbouring voxels to make a pattern. Non-inner voxels are the rest, i.e., those that are close to the boundary and therefore do not have as many neighbouring voxels around them. How far away from the boundary a voxel has to be in order to be an inner voxel depends on how the neighbourhood of voxels are defined.

Several parameters are needed to specify the dimensions of neighbourhoods and these are collected in a special type of structure called Neighbor-Mask. It contains the following parameters:

mat: 3D integer array of the shape of a neighbourhood of an inner voxel. Voxels in positions where the element of mat is 1 are included in the neighbourhood, and voxels where the corresponding element is 0 are not included. The value corresponding to the center voxel does not matter.

nc, mc, pc: integers denoting the coordinates of the center voxel in the mat array. (The origin of a pattern is its bottom left upper corner.)

n, m, p: size of the patterns, i.e., the mat array includes $n \times m \times p$ voxels.

nodes: 2D integer array with relative coordinates from the center voxel to each of its neighbours. That means, given the image coordinates of a voxel, nodes can be used to compute the image coordinates of all of the neighbours of the voxel. The array has a row for each neighbour in a pattern and three columns that holds the coordinates in each dimension.

Neighbourhoods can be defined in two ways: 1) If the patterns have the simple shape of a hyper-rectangle, the user can simply specify the size of the patterns, m, n and p. These must be odd numbers and the center voxel of the pattern is then assumed to be directly in the middle. This is the most common type of pattern to use. 2) If the patters are not that simple, the user can directly specify the mat array and its center coordinate (nc, mc, pc). The size of the mat array do not have to be odd, and the center can be located anywhere.

The parameter sN is used throughout the procedures as the number of voxels in a pattern excluding the center voxel. This means that nodes has sN rows. In the procedures the structure specifying the neighbourhoods is called Nmask.

Figure 2 shows a tiny example of a training image and a neighbourhood that could be used to describe the multiple-point statistics of it. To define this non-rectangular neighbourhood the user will need to specify the mat array:

$$
\mathsf{Nmask\%mat} \;=\; \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},
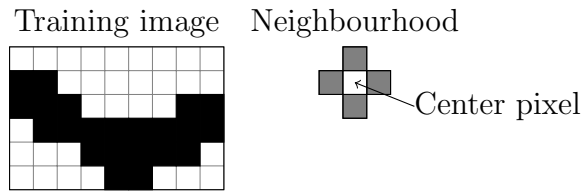$$

9

Training image   Neighbourhood

Figure 2: Tiny training image used for illustration purposes. A neighbour-hood is chosen to be the (up to) four nearest neighbours as shown on the right of the figure. The pixels within the neighbourhood are numbered according to their row, column and then layer index. How to define this neighbourhood is explained in the text. The resulting tree can be seen in Figure 4.

.

and its center coordinates:

$$Nmask\%nc = 2,$$
$$Nmask\%mc = 2,$$
$$Nmask\%pc = 1.$$

Then the FMM procedure derives the size of the patterns:

$$Nmask\%n = 3,$$
$$Nmask\%m = 3,$$
$$Nmask\%p = 1,$$

leading to the following relative row, column and layer coordinates of neigh-bours:

$$Nmask\%nodes = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Keep in mind, that even for 2D images the third dimension does exist, it will just have the size one.

2.6 The Tree Structure

A tree is a complex structure and most vital for the computational feasi-bility of the implementation. The purpose of the tree is to store the patterns

of a training image and their counts, and then to use them to describe the multiple-point statistics of the training image.

The tree is implemented as a so-called linked list. This is the same approach that was applied in the SNESIM algorithm (Strebelle, 2002) when creating search trees in order to overcome the problems of generating and storing large data bases of patterns. In our implementation trees are furthermore used to easily generate frequency distributions, which is the actual input to the dissimilarity function c from Equation (4).

We like to think of trees as consisting of a set of nodes and edges as a tree in the mathematical sense of a graph. A tree, T, is based on a root node, and it is often that node we pass along the different procedures. From this root node we can navigate deeper into the tree via links, which in this case are pointers to the next nodes.

A node of a tree is defined as a structure $\mathsf{streenode}$ containing three variables:

depth: integer defining how deep into the tree this node is located.

repl: real array with $\mathsf{sV}{+}1$ elements. The array holds the count of patterns in the image that have a certain partial pattern and center value $0, 1, \ldots, \mathsf{sV}$ respectively. The partial pattern is dependent on the depth of the root, and the deeper into the tree a node is located, the more voxels of the patterns are used in the partial pattern.

next: array of pointers with $\mathsf{sV}{+}1$ elements, these are the links to nodes placed a level deeper into the tree.

Figure 3 shows the structure defined to describe a node. A tree of an image is constructed by first creating the root node and then adding new nodes as the image is being scanned and new patterns found. Assigning the root node depth level 0, the maximum depth of a tree is sN, and at any depth i there can maximum be $(\mathsf{sV}{+}1)^i$ nodes.

Let T denote the root node of a tree; this node contains information of the center values themselves, not including any neighbouring voxel values. We define the $\mathsf{T\%repl}$ array to hold the unscaled distribution of voxel values of inner voxels in the image, so simply the counts of how many inner voxels in the image have each of the values $0, 1, \ldots, \mathsf{sV}$. $\mathsf{T\%repl(k)}$ holds the count of voxels with value $k{-}1$.
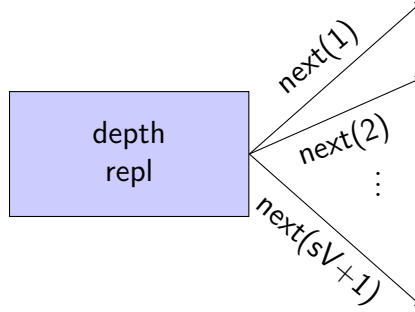
Figure 3: Illustration of a node of a tree as represented by the streenode structure.

The pointers in the array T%next point to sV+1 new nodes of the tree. These nodes are at depth level 1 and they therefore contain information about center voxels taking into consideration the value of their first neighbouring voxel. (The neighboring voxels are ordered according to their voxel index.) As the first neighbouring voxel can have sV+1 different values we need the same number of pointers to cover all cases. This means, the pointer T%next(i) points to the node representing all partial patterns, where the first neighbouring voxel has value $i-1$.

The unscaled distribution for all values of center voxels will be stored in the repl array of that node. It can be accessed by T%next(i)%repl. This means the element T%next(i)%repl(k) holds the counts of patterns in the image where the first neighbouring voxel has the value $i-1$ and the center voxel has value $k-1$. In the same manner, the jth pointer of this node, T%next(i)%next(j), points to the node of patterns where the values of the two first neighbouring voxels are $i-1$ and $j-1$, respectively. The node holds the unscaled conditional probability distribution of the value of a center voxel given these specific values of the first two neighbouring voxels. This is repeated until depth level sN, where all sN neighbouring voxels have been included in the partial structure, that each node represents.

To sum up, an arbitrary node T provides us the following information:

T%depth: depth level in the tree where the node is placed.

T%repl(k): count of a specific partial pattern in the image with center value $k - 1$. The partial pattern is unknown to this node, but the values of

12

the first T%depth neighbouring voxels are given by the location of the node in the tree.

T%next(i) pointer to the node with the same partial pattern as the current and where the T%depth+1th voxel has value $i - 1$.

Notice how a node T cannot give us any information about the partial patterns it represents. From a node we can only extract information that are deeper into the tree and not information that belong to a previous level.

The frequency distribution of an image is the (unscaled) distribution of patterns. As the bottom level of the tree contains exactly the counts of patterns with all of the possible combination of center and neighbourhood voxel values, the frequency distribution can simply be constructed by combining all existing repl arrays at depth level sN.

Recall that Figure 2 shows an example of a training image and an example of a neighbourhood. This is a very tiny training image and neighbourhood chosen for illustration purposes only. The resulting tree is seen in Figure 4. The frequency distribution of the image is constructed by extracting all 4th level repl arrays of the tree.

On each node is written the values of its repl array, and on each edge is written the colour of the neighbouring voxel represented by the current depth level. Black voxels are assigned the value 0 and white voxels have the value 1.

To compare an image to a training image a tree describing its multiple-point statistics must be derived and its frequency distribution determined. This allows for the evaluation of c. However, constructing the tree is done in a different manner than for a training image itself. When constructing the tree we take advantage of the fact that the dissimilarity function c depends only on the patterns of the image that also appear in the training image. We therefore generate not the tree containing all patterns found in the image but only those patterns that are also found in the training image. That means, the tree of the image will have the same shape (same nodes and edges) as the tree of the training image. This makes the frequency distribution consisting of all bottom level repl arrays directly comparable to the frequency distribution of the training image, as they, element by element, describe the count of identical patterns in the two images.

Non-inner voxels of an image that is not a training image are treated slightly different than non-inner voxels of a training image. They contribute
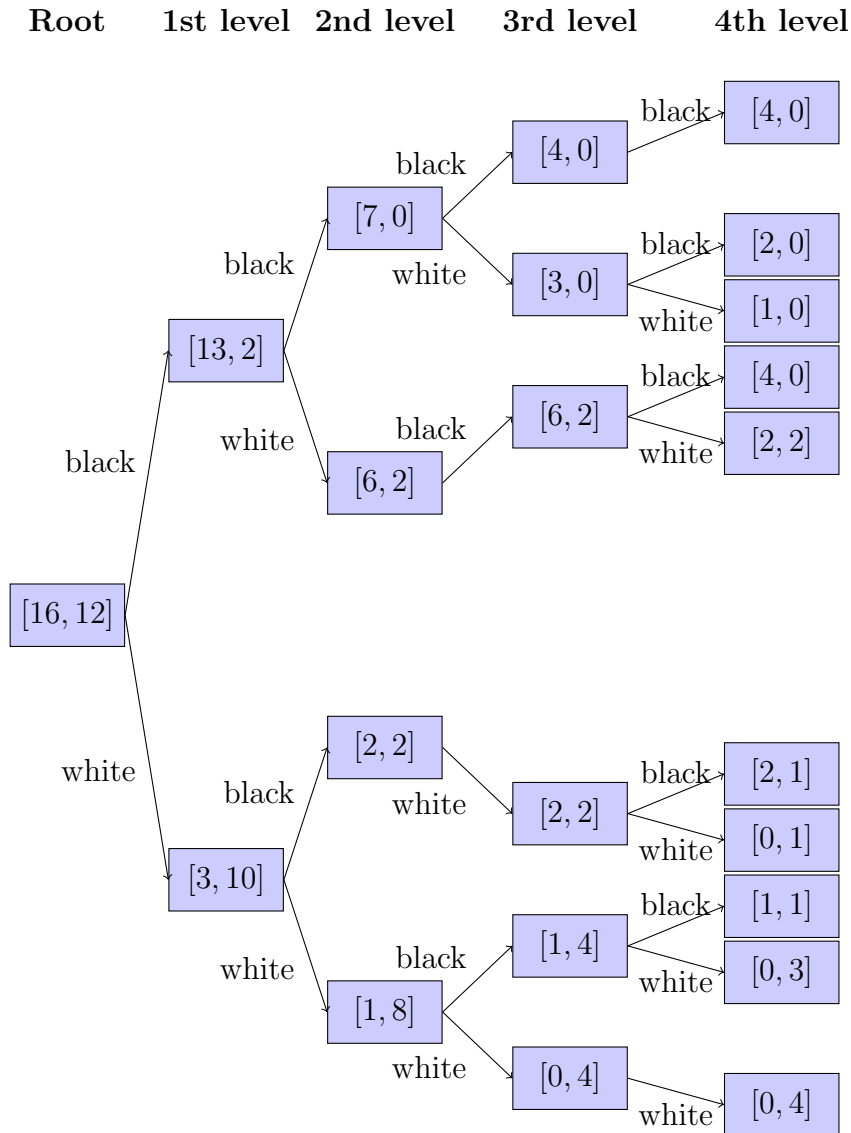
Figure 4: The tree structure of the tiny training image in Figure 2 using a neighbourhood consisting of the four closest voxels. We assign black voxels the value of 0 and white voxels the value of 1. Notice how the count of each pattern is represented in the tree. For instance, the number of black pixels with all black neighbouring pixels is 4. This is seen by starting in the root node, following all edges labelled black until reaching the bottom level, and then accessing the first element of the repl array. Also notice how for every single node, if you sum the repl arrays of the nodes it is pointing to, you get the repl array of the node itself. This means all repl arrays only hold counts of inner pixel.

14

to the tree like inner voxels, but they are typically represented by multiple patterns. They contribute with a total count of 1, like the inner voxels, and the count for each type of pattern is proportional to their marginal conditional probability density from the tree of the training image.

2.7   Perturbation Domain

The optimization problem defining the FM model from Equation 3 is solved using an iterative solution method that searches through the model space. The search is carried out by visiting new images that are defined as perturbations of current images. They are created by perturbed the voxel values in a certain domain of an image.

A set of parameters is needed to specify the size of the domain of an image that then needs to be erased and re-simulated to create a perturbed image. The domain is assumed to be hyper-rectangular. To define it we have the DomainMask structure that contains the following parameters:

n, m, p: integers, dimensions of the domain to be re-simulated. These must be odd numbers.

nodes: 2D integer array with relative coordinates from the center voxel of the domain to each other voxel in the domain.

mat: 1D integer array with sN elements. This array holds a distance from the center voxel of a neighbourhood to each of its neighbouring voxels. It is used in the re-simulation to determine on which voxels the simulation should be conditioned.

The domain structure is Dblock. Like for the neighbourhood mask the user does not need to specify most of its parameters. In fact, one should only decide on the dimensions of the block, Dblock%n, Dblock%m, Dblock%p, and the FMM procedure then constructs the remaining. For the distance array is used the L_1-norm.

2.8   Optimization Options

Simulated annealing is used as the solution method to the optimization problem defining the $\mathbf{m}^{\mathrm{FM}}$, and to hold the parameters used by the algorithm

we have the type option. It holds the following parameters:

t0: real number, the initial temperature.

tmin: real number, the final temperature.

maxIter: integer, maximum number of iterations allowed to be used per voxel parameter.

runs: integer, number of times to run the simulated annealing algorithm.

multigrid: integer, number of multigrids to use.

condopt: logical, in case of multiple grids used, it determines if the solution on a fine grid should be conditioned on the optimal solution from the coarser grid (condopt = .true.) or not (condopt = .false.).

The simulated annealing uses an exponential cooling rate that is calculated such that the maximum number of iterations allowed is exactly the number of iterations used. The implementation of the FM method has been prepared for multiple grid simulation but these are not yet implemented. This can be done by implementing a loop in the FMM procedure such that CompOptimalImage will be called with different grids. Also in some cases it can be beneficiary to restart the simulated annealing algorithm and although this option has not yet been implemented the code has been prepared. A loop can be inserted in the CompOptimalImage procedure so that the simulated annealing scheme is run multiple times.

3 Example: Crosshole Travel Time Tomography

As an example of how to use the Frequency Matching method we will show how to solve a synthetic crosshole travel time tomography problem similar to the one described in Lange et al. (2012). Crosshole travel time tomography involves the measurement of seismic travel times between two or more boreholes in order to determine an image of seismic velocities in the intervening subsurface. Seismic energy is released from sources located in one borehole and recorded at multiple receiver locations in another borehole.
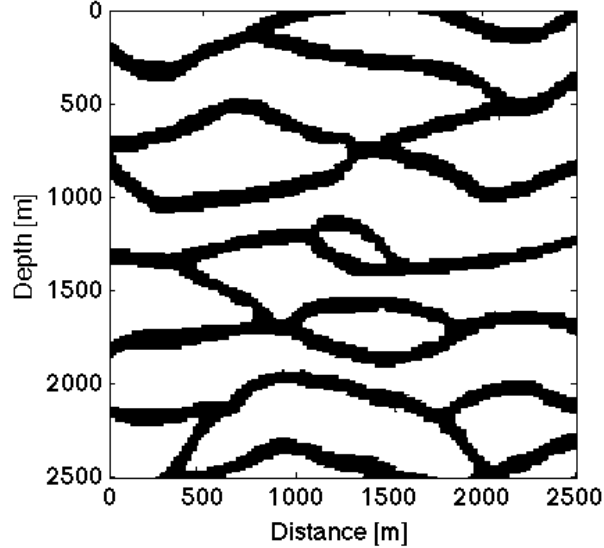
Figure 5: Training image (size: 250 by 250 pixels).

In this way a dense tomographic data set that covers the interborehole region is obtained.

We create a synthetic test case based on a setup with two vertical boreholes. The horizontal distance between them is 500 meters and they each have a depth of 500 meters. The two-dimensional vertical domain between the boreholes is divided into 120 times 50 quadratic cells. The seismic velocity is assumed constant within each cell. The model parameters of the problem are these propagation speeds, meaning the problem has 6000 unknown model parameters. The observed data is the recorded first arrival times from the seismic signals. In each borehole are placed 12 equally distributed sources and 48 equally distributed receivers. We assume a linear relation between the data observations and the model parameters. The sensitivity of a seismic signal is simulated as straight rays.

It is assumed that the inter-borehole region consists of a background with slow propagation speed and a horizontal channel structure of zones of high propagation speed. The speeds are chosen as 1600 meters per second and 2000 meters per second, respectively. The a priori knowledge of the channel structure is assumed described by the training image in Figure 5. We have chosen to let the neighbourhood of a pixel consist of its 36 closest neighbours

17
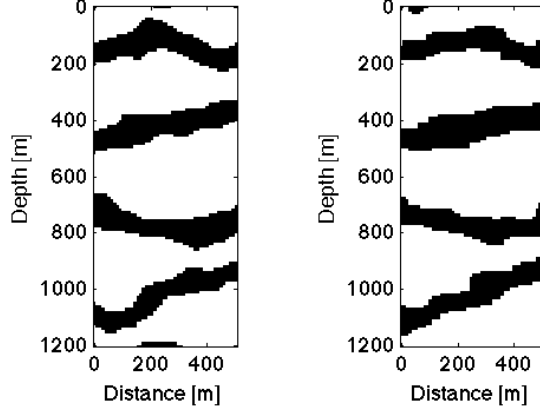
(a) Reference model.   (b) FM model.

Figure 6: Reference model for the synthetic crosshole travel time tomography example and its computed FM model. The size of the models ares 120 by 50 pixels.

specified by:

$$
\mathsf{Nmask\%mat} \;=\;
\begin{bmatrix}
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0
\end{bmatrix} .
$$

A reference model is generated based on the training image using the SNESIM (Strebelle, 2002) algorithm. The first arrival times of the reference model is simulated. These are then added 5% relative Gaussian noise and assumed to be the observed data. Figure 6a shows the reference model.

The forward problem   Voxels belonging to the background are assigned to the category 0 and voxels belonging to the zones of high propagation speed are assigned the category 1. The forward problem is linear in the inverse of

the propagation speeds, i.e., the physical values of the voxel values of the two categories are specified as $\mathsf{cat} = [1/1600, 1/2000]$.

The observed data and the coeffecient matrix from the forward problem is generated using MATLAB. They have then been stored in a text file that can be read into Fortran using the standard read routine and then saved in the parameters dobs and Gmat, respectively. The problem has 1152 data observations and 6000 model parameters so it is severely under-determined.

The % noise added to the reference model is independent and has estimated standard deviation $\hat{\sigma} = 1.9 \cdot 10^{-2}$. This yields the data covariance matrix $\mathbf{C_d} = \hat{\sigma}^2 I$, where I is the identity matrix. The inverse data covariance matrix invCov is then defined as the inverse of this.

The prior term   The weighting constant multiplied to the prior term in Equation (3) is chosen as $\alpha = 10^{-1}$, which means $\mathsf{alpha} = 10^{-2}$.

Perturbation of images   The domain of voxels to be re-simulated when creating a perturbed image is chosen as $\mathsf{nD} = 13$, $\mathsf{mD} = 13$ and $\mathsf{pD} = 1$.

Optimisation parameters   The cooling rate is defined by the starting temperature $\mathsf{t0} = 10^2$ and the minimum temperature $\mathsf{tmin} = 10^{-5}$. The simulated annealing algorithm is allowed to use $\mathsf{iter} = 0.5$ for each of the 6000 pixels in the image.

Computing the optimal model using the allowed 3000 iterations took approximately 16 minutes on a Macbook Pro 2.66 GHz equipped with an Intel Core 2 Duo processor and 4 GB of RAM.

The computed optimal model is shown next to the reference model in Figure 6. It is seen how it correctly locates the channels. The width and curvature of the channels also clearly resembles those of the reference model. We therefore conclude that the choice of weighting constant alpha and the optimisation parameters were suitable for the problem at hand. The example successfully illustrates how the Fortran implementation of the Frequency Matching method can be applied.

4 Bibliography

Guardiano, F., Srivastava, R. M., 1993. Multivariate geostatistics: Beyond bivariate moments. Geostat-Troia 1, 133–144.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., 1983. Optimization by simulated annealing. Science 220 (4598), 671–680.

Lange, K., Frydendall, J., Cordua, K. S., Hansen, T. M., Melnikova, Y., Mosegaard, K., 2012. A frequency matching method: Solving inverse problems by use of geologically realistic prior information. Mathematical Geosciences, 1–2110.1007/s11004-012-9417-2.
URL http://dx.doi.org/10.1007/s11004-012-9417-2

Sheskin, D., 2004. Handbook of Parametric and Nonparametric Statistical Procedures. Chapman & Hal/CRC, pp. 493–500.

Strebelle, S., 2002. Conditional simulation of complex geological structures using multiple-point statistics. Mathematical Geology 34, 1–21.

Tarantola, A., 2005. Inverse Problem Theory and Methods for Model Parameter Estimation. SIAM.

List of Appendices

A Primary Procedures

The following is a list of the primary Fortran procedures in the FM implementation. The purpose of each procedure is briefly explained, and any non-trivial or otherwise interesting details in the implementation are discussed. Also a list of input and/or output variables is provided. The lists hold the variable name, a short description of its use and its type.

To see which procedures call others we refer to Figure 1 and the discussing of it in the text. The procedures are listed in the order they are called which also appears from the figure. Auxiliary procedures are listed in B.

A.1 FMM

This procedure acts as an intermediary between the user specified input parameters and the implemented FM method. It reads the multiple-points statistics from the training image and sets up all the necessary inputs for the simulated annealing scheme based on the user inputs.

The procedure calls the CompOptimalImage procedure to compute the FM model (3). This model along with its frequency distribution, the frequency distribution of the training image and other parameters of special interest are written to files. These can later to loaded into for instance MATLAB to visualize the results.

In case the code is modified to handle multiple grids this would be a suitable procedure in which to loop over the grid levels, and for each level set up the corresponding tree of the training image and do the conversion from coarse to fine grid before calling the CompOptimalImage.

Variable	Description	Type
Z0	Initial image used as starting image for the iterative solution method. This image should satisfy hard data constraints, if any.	3D integer array
Ztrain	Training image.	3D integer array

Zcond	This array has the same dimensions as the image Z0, and it is used to state if any of the voxels should satisfy hard data constraints. If there are no hard data constraints the array should be all false. If some voxels are only allowed to take on a specific value the corresponding element of Zcond should be true.	3D logical array
Nmask	Contains the parameters that define neighbour-hoods.	structure
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
InvProb	Parameters specifying the linear inverse problem.	structure
alpha	Weighting parameter α^2 of the prior term in the objective function.	real
Dblock	Contains the parameters controlling the perturbation of an image.	structure
options	Parameters for the simulated annealing scheme.	structure
solutions	Structure holding the results from the simulated annealing. These are written to files.	structure
Ps	Holds the values of each of the two terms in the objective function for each iteration of the simulated annealing algorithm.	2D real array

A.2   InferTrainTree

This procedure generates the tree, Ttrain, describing the multiple-point statistics of a training image, Ztrain. Patterns are extracted one by one from the training image and added to the tree.

Variable	Description	Type
Ztrain	Training image.	3D integer array
nodes	Array from the Nmask structure.	2D integer array

sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer
Ttrain	Tree of patterns extracted from the training image.	tree

A.3 Tree2Hist

This procedure constructs the frequency distribution (or the histogram), H, of an image given its tree, T. The frequency distribution is a two dimensional array with sV+1 rows and a column for each combination of voxel values in a neighbourhood. The ith row has the count of the appearances of the different neighbourhoods with center voxel having the value $i-1$.

This format has the advantage that each column of the frequency distribution is the unscaled conditional probability distribution of the value of center voxel given the values of its neighbouring voxels. This makes the conditional distributions easily assessable; they are used, for instance, for re-simulating voxel values. The disadvantage is that we might store more zero elements than necessary although no more than sV times too many.

Variable	Description	Type
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer
T	Tree of patterns extracted from an image Z.	tree
H	Frequency distribution of the image Z.	2D real array

A.4 CompOptimalImage

This procedure is the most central in the implementation of the FM method as it is the one that solves the inverse problem by use of the simulated annealing algorithm. It takes multiple inputs from the starting image and the multiple-point statistics learned from the training image to the FM parameters specifying the neighbourhoods and the parameters associated with simulating perturbed images. It returns the computed optimal image, $\mathbf{m}^{\text{FM}}$, as well as its frequency distribution. To check the convergence of the simulated annealing algorithm it also returns the objective function values for all iterations.

Variable	Description	Type
Z0	Initial image used as starting image for the iterative solution method. This image should satisfy hard data constraints, if any.	3D integer array
Zcond	This array has the same dimensions as the image Z0, and it is used to state if any of the voxels should satisfy hard data constraints. If there are no hard data constraints the array should be all false. If some voxels are only allowed to take on a specific value the corresponding element of Zcond should be true.	3D logical array
Ttrain	Tree of patterns extracted from the training image.	tree
Htrain	Frequency distribution of the training image.	2D real array
nodes	Array from the Nmask structure.	2D integer array
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer
InvProb	Parameters specifying the linear inverse problem.	structure
alpha	Weighting parameter α^2 of the prior term in the objective function.	real

Dblock	Contains the parameters controlling the perturbation of an image.	structure
options	Parameters for the simulated annealing scheme.	structure
Zopt	The optimal image, the $\mathbf{m}^{\mathrm{FM}}$, computed by the simulated annealing algorithm.	3D integer array
Hopt	Frequency distribution of the optimal image Zopt.	2D real array
Ps	Holds the values of each of the two terms in the objective function for each iteration of the simulated annealing algorithm.	2D real array

A.5   InferTree

This procedure infers the tree of an image so that its multiple-point statistics can be compared to those of a training image. It takes as input the tree of the training image that the image should be compared to. This is necessary as the tree should only contain patters also found in the training image. Once the image has been scanned and all patterns that should be stored has been added to the tree, the procedure shapeTree is called, to ensure that the tree of the image has the same shape as the tree of the training image, which is needed in order to easily compare their frequency distributions.

Variable	Description	Type
Z	Image.	3D integer array
nodes	Array from the Nmask structure.	2D integer array
Ttrain	Tree of patterns extracted from the training image.	tree
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer

T	Tree of patterns extracted from the image Z.	tree
Zex	This array is the same size as the image Z and it is used to indicate which voxels of the image are center of patterns found also in the training image. It is used by the procedure getNewIt to propose where an image should be perturbed.	3D logical array

A.6 CompObjFun

This procedure computes the values of each of the terms in the objective function from Equation (3). To be able to track convergence of the simulated annealing algorithm the terms are not added but instead the procedure returns a two-element array P such that:

$$
\begin{aligned}
\mathsf{P(1)} &= \frac{1}{2}\|\mathbf{d}^{\mathrm{obs}} - \mathbf{Gm}\|_{\mathbf{C_d}}^2, \\
\mathsf{P(2)} &= \alpha^2 c(\mathbf{m}, \mathbf{m}^{TI}).
\end{aligned}
$$

The dissimilarity function c is evaluated by the CompChiDist procedure and the data misfit is computed by the CompDataFit procedure.

In case the inverse problem is not linear the CompDataFit procedure needs to be replaced by an implementation of the non-linear forward mapping.

Variable	Description	Type
H	Frequency distribution of the image Z.	2D real array
Htrain	Frequency distribution of the training image.	2D real array
N	Number of voxels in the image Z.	integer
Z	Image.	3D integer array
InvProb	Parameters specifying the linear inverse problem.	structure
alpha	Weighting parameter α^2 of the prior term in the objective function.	real
P	Two-element array that holds the values of each of the terms in the objective function.	1D real array

27

A.7 CompChiDist

This procedure is called by CompObjFun and computes the dissimilarity of an image compared to a training image by computing the distance between their frequency distributions.

Variable	Description	Type
H	Frequency distribution of the image Z.	2D real array
Htrain	Frequency distribution of the training image.	2D real array
N	Number of voxels in the image Z.	integer
X	Value of the dissimilarity function of the two frequency distributions.	real

A.8 CompDataFit

This procedure is called by CompObjFun and computes the data misfit of a model.

Variable	Description	Type
Z	Image.	3D integer array
InvProb	Parameters specifying the linear inverse problem.	structure
L	Value of the data misfit of for the image	real

A.9 SimNewImage

This procedure is used to perturb images. Provided with the current image from the simulated annealing iteration, Ztest, it will return a new, perturbed image, Znew. Znew is generated by erasing the values of a subset of the voxels and then re-simulating them using sequential simulation conditioned on the voxel values of the remaining part of the image.

This procedure takes as input the row, column and layer index of a voxel. The voxel is the center of a domain where voxel values are erased. The values of the voxels in the domain are then re-simulated one by one conditioned on the values of all voxels outside the domain and the already re-simulated values inside the domain. Of course voxels that should satisfy hard data constraints are not allowed to be changed and these are therefore not erased and re-simulated. Their values are kept and instead used to condition the re-simulation on.

The re-simulated values voxels are stored separately. The perturbed image is initially set identical to the original image. Its voxel values are then changed one at a time until all the voxels in the re-simulated domain has been updated. This allows the tree of the original image to be updated to the tree of the new, perturbed image. Tnew is initially set to be an exact copy of Ttest, and it is then updated iteratively for every voxel that has been assigned a new value. This way of iteratively updating the tree is much cheaper than generating the tree of the perturbed image from scratch.

Once the perturbation of the image and the updating of its tree is completed the frequency distribution can be computed from the tree. SimNewImage of course only updates the tree and recomputes the frequency distribution if the perturbed image is in fact different from the original image.

Variable	Description	Type
Ztest	Current image.	3D integer array

Zcond	Array used to specify which voxels are subject to hard data constraints. The values of such voxels are known and the voxels are used to conditioned upon in the simulation of other voxel values. The simulation algorithm is not allowed to erase and re-simulate the values of voxels that are conditioned upon. Instead these values are considered known and should be used when re-simulating other voxel values.	3D logical array
Zex	This array is the same size as the image Ztest and it is used to keep track of if the pattern that a voxel is the center of exists anywhere in the training image or not. It is used by the procedure getNewIt.	3D logical array
Ttest	Tree of patterns in the image Ztest.	tree
Ttrain	Tree of patterns extracted from the training image.	tree
i, j, k	Indices in the image of the voxel that is chosen such that the image is perturbed by erasing and the re-simulating the values of voxels in a domain around it.	integers
nodes	Array from the Nmask structure.	2D integer array
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer
Dblock	Contains the parameters controlling the perturbation of an image.	structure
Znew	Perturbed image based on the current image Ztest.	3D integer array
Zexnew	Similar to Zex, this array denotes which voxels in the image Znew are centres of patterns also found in the training image.	3D logical array
Tnew	Tree of patterns in the image Znew.	tree
Hnew	Frequency distribution of the perturbed image Znew.	2D real array

newImage	Indicates whether the perturbed image Znew is different from the original image Ztest (newImage = .true.) or not (newImage = .false.).	logical

A.10   SimVoxel

This procedure is needed to simulate the value of a voxel. It extracts from Ttrain the (unscaled) conditional probability distribution of the value of a center voxel given the values of the neighbouring voxels. It returns this conditional probability distribution and the voxel can then be assigned a value drawn from it.

In case the multiple-point statistics of Ttrain does not allow for this occurrence of the values of the neighbourhood voxels, i.e., no patterns of the training image matches the partial pattern, the voxels will be dropped one by one from the conditioning until a conditional probability distribution can be extracted. It is always the voxel furthest away from the center voxel that will be dropped. Dropped voxels are assigned the value -1 and they then appear as unknown.

If all neighbouring voxels are dropped the unconditioned distribution of voxel values in the training image will be used as the unscaled probability distribution of voxels also in the image.

Variable	Description	Type
Zvec	Holds the values of voxels in a neighbourhood. Unknown voxel values are assigned a value of -1.	1D integer array
Ttrain	Tree of patterns extracted from the training image.	tree
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer
D	Part of the Dblock structure. Holds the distances from each of the neighbouring voxel to the center voxel.	1D integer array

Variable	Description	Type
hc	Holds the unscaled probability distribution of the value of the center voxel conditioned on its neighbouring voxels.	1D real array

A.11   UpdateSA

The point of this procedure is solely to simplify the code. It copies an image and its associated variables into another set of variables. This is for instance used in the simulated annealing scheme when a new image is accepted. Then the procedure is used to copy the new image Znew into the variable of the current image Ztest and to update its associated variables such that they contain the variables Tnew, Hnew etc. The updating of the variables is trivial except for the tree. The updating of the tree is done by the CopyTree procedure.

Variable	Description	Type
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer
Zin	The image that should be copied.	3D integer array
Zinex	Array to determine which patterns in Zin exist in the training image.	3D logical array
Tin	Tree of the image Zin.	tree
Hin	Frequency distribution of the image Zin.	2D real array
Pin	Array with the value of each term of the objective function of the image Zin.	1D real array
Zout	The image variable into which Zin should be copied.	3D integer array
Zexout	Array to determine which patterns in Zout exist in the training image.	3D logical array
Tout	Variable that holds the tree of the image Zout. This variable should be updated to hold Tin.	tree

Hout	Variable that holds the frequency distribution associated with the image Zout, this variable should be updated to hold Hin.	2D real array
Pout	Array with the values of the terms of the objective function associated with the image Zout, this variable should be updated to hold Pin.	1D real array

B   Auxiliary Procedures

The following list describes the auxiliary procedures of the FM implementation. These are not of great importance to understand the code but necessary building blocks.

B.1   getNewIt

The procedure determines which part of an image should be re-simulated in order to compute a perturbed image. It returns the row, column and layer index of the voxel that is the center of the domain, which should be re-simulated.. To reduce the number of iterations needed for the simulated annealing algorithm to converge, we wish to choose a voxel that will result in maximal change to the perturbed image.

Consider two different voxels in the image. Assume that in an area around the first voxel the image looks very similar to the training image. Erasing the voxel values in a domain around in this area and re-simulating them based on the multiple-point statistics of the training image will then likely result in a perturbed image that is very similar to the original image. There is no reason to expect the perturbed image to be a significantly better solution to the inverse problem, than the image was before perturbing it.

Now assume that the original image in an area around the second voxel looks very different from anything seen in the training image. Erasing and re-simulating the voxel values in a domain centred in the second voxel will then create a very different perturbed image. This perturbed image is much more likely to be a better solution to the inverse problem as it will fit the multiple-points statistics of the training image better. In case is not a better data fit, it could potentially belong to a different, unexplored part of the

33

model space. While the image has areas that are dissimilar to any area of the training image, we would like these areas to have a high relative probability to be re-simulated.

The procedure goes through randomly proposed voxels and picks them with a probability that is proportional to the number of undesirable patterns in their neighbourhood (a pattern is deemed undesirable if it does not exist in the training image).

The procedure suggest a random voxel and then scans the neighbourhood voxels, say it has y neighbouring voxels. It counts how many of these, including the voxel itself, are centrers of undesirable patterns, let us denote that number x. The suggestd voxel is then accepted as a center for the perturbation domain with probability:

$$\text{Prob}(\textsf{voxel}) \quad = \quad \frac{x}{y+2}$$

The denominator $y + 1$ comes from the number of patterns in the neighbourhood plus the pattern from the voxel itself. The extra $+1$ is added such that areas with no undesirable patterns, i.e., $x = y$, have a small yet non-zero probability to be chosen. Otherwise the iterative algorithm might be stuck and prevented to converge, as an image can have no undesirable patterns without matching the frequency distribution of the training image and without matching the data fit.

Variable	Description	Type
Zex	Denotes which voxels in the image are centres of patterns found also in the training image. An element of Zex is true if the pattern centred in the corresponding voxel exists in the training image. And contrary, an element of Zex is false if the corresponding voxel of the image is center of a pattern not found in the training image.	3D logical array
nodes	Array from the Nmask structure.	2D integer array
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of sN + 1 voxels.	integer
i, j, k	Indices of the three dimensions of the image for the voxel that is chosen as the center of the domain to be re-simulated.	integers

B.2 getNeighborhood

Given an image Z and a row, column and layer index of a voxel in the image, this procedure extracts the voxel values of the neighbouring voxels. The extracted voxel values will be flattened into a 1D array.

Variable	Description	Type
Z	Image.	3D integer array
i, j, k	Indices in the image of the center voxel of the neighborhood.	integers
nodes	Array from the Nmask structure.	2D integer array
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer
Zvec	Holds the values of voxels in a neighbourhood. Unknown voxel values are assigned a value of -1.	1D integer array
innervoxel	Denotes if the voxel with the specified indices was an inner voxel (innervoxel = true) or not (innervoxel = false)	logical

B.3 getCPDF

This is a recursive procedure used by SimVoxel to compute the conditional probability density function of the value of a voxel given the values of its neighbouring voxels.

The procedure searches through the tree (depth first) for patterns that matches the neighbourhood values in Zvec and adds up the counts of patterns depending on the value of their center voxel. cpdf is then a sV+1 element array with the counts of patterns matching the values of the neighbourhood voxels for each of the sV+1 possible value of the center voxel.

Variable	Description	Type
Ttrain	Tree of patterns extracted from the training image.	tree
Zvec	Holds the values of voxels in a neighbourhood. Unknown voxel values are assigned a value of -1.	1D integer array
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer
cpdf	Current counts of patterns matching the values in Zvec.	1D real array

B.4 ExtendTree

This procedure adds another node to a tree. It takes as input a tree node where the pointers in the next array are not yet associated. The procedure initialises them by allocating their repl arrays and setting the counts to 0. It also sets their depth values to be one deeper than the current T%depth, and it nullifies their next arrays.

Variable	Description	Type
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
T	Tree to be extended	tree

B.5 ShapeTree

The procedure shapes a tree T such that it has the same shape as the Ttrain, based on which it was constructed. By construction the tree cannot be

any bigger than Ttrain, as it holds only patterns that are also found in the training image. However, it can be smaller, as some patterns may appear in the training image but not in the image from which T is constructed. The nodes representing such patterns need to be added with the appearance count of zero.

Ensuring the trees have the same shape simplifies future operations such as updating of trees and comparison of frequency distributions.

Variable	Description	Type
T	Tree of patterns extracted from an image Z.	tree
Ttrain	Tree of patterns extracted from the training image.	tree
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer

B.6   CopyTree

As trees are complex structures we cannot just copy the content of one tree, Told, into another tree, Tnew, in the way we usually do with arrays. Simply saying Tnew = Told is not defined and therefore has no meaning. The procedure is therefore needed whenever we need to make a copy of a tree. It is used, for instance, by the UpdateSA procedure to copy the tree of the perturbed image Tnew into the variable holding the tree for the current image Ttest.

The procedure initialises a new tree from scratch and then copies the content from the old tree into it node by node, without overwriting Told.

Variable	Description	Type
Told	Tree of patterns extracted from an image.	3D integer array

sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer
Tnew	Tree, which is a copy of Told.	3D integer array

B.7 DeallocateTree

Recursive procedure that deallocates a tree. This is done by deallocating the repl array and the next array associated with each node for all nodes, one node at a time.

Variable	Description	Type
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
T	Tree node to be deallocated.	tree

B.8 UpdateTrainTree

This recursive procedure is used to add a pattern to the tree of the training image. For each pattern, InferTrainTree calls the procedure that then recursively calls itself while going deeper and deeper into the tree. This causes the count of the pattern to be added all the way to the bottom of the tree. If a type of pattern has not already been added to the tree the procedure ExtendTree is used to extend the tree with extra nodes before the pattern can be added.

Variable	Description	Type
Ttrain	Current tree node	tree

Variable	Description	Type
Zvec	Holds the values of the voxels in a neighbourhood.	1D integer array
cv	Voxel value of the center voxel of the pattern.	integer
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer

B.9 wrapUpdateTree

This procedure is called by InferTree to add a pattern to the tree when the center of the pattern is an inner voxel. It works as a wrapper for the recursive UpdateTree.

Variable	Description	Type
T	Tree of patterns so far extracted from the image Z.	tree
Ttrain	Tree of patterns extracted from the training image.	tree
Zvec	Holds the values of voxels in a neighbourhood.	1D integer array
cv	Voxel value of the center voxel of the pattern.	integer
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer
exist	Indicate whether the pattern is found in the training image (exist = true) or not (exist =false).	logical

B.10 UpdateTree

Recursive procedure that adds the contribution from a pattern which center voxel is an inner voxel. It goes through the pattern voxel by voxel, and adds it contribution node by node as deep into the tree as allowed. Recall that the shape of the tree must not be changed as it shall remain the same as the shape of the tree of the training image. Therefore patterns from the image that are not found in the training image will not contribute to any counts after a certain level of depth as the nodes representing them do not exist. This also means they do not appear in the frequency distribution.

The inputs of the procedure is the same as of wrapUpdateTree.

B.11 wrapUpdateTreeBoundary

Like wrapUpdateTree this procedure is called by InferTree and it works as a wrapper for UpdateTreeBounday. The procedure is used to add a pattern which center is not an inner voxel.

Variable	Description	Type
T	Tree of patterns so far extracted from the image Z.	tree
Ttrain	Tree of patterns extracted from the training image.	tree
cpdfold	Marginal conditional probability distribution of the value of a center voxels conditioned on the values of the neighbouring voxels in Zvec.	1D real array
Zvec	Holds the values of the neighbouring voxels in the pattern. Imaginary voxels have been assigned the value -1.	1D integer array
cv	Voxel value of the center voxel of the pattern.	integer
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer

exist	Indicate whether the pattern is found in the logical training image (exist = true) or not (exist =false).

B.12 UpdateTreeBoundary

Recursive procedure that adds the contribution from a pattern which center voxel is not an inner voxel. The contribution of the pattern is set to be proportional to the marginal conditional probability of the center value given the voxel values of the neighbouring voxels. This distinction between inner voxels (handled by UpdateTree) and non-inner voxels (handled by UpdateTreeBoundary) is necessary as they contribute differently to the tree.

   UpdateTreeBoundary goes through the tree. For each level it assigns the contribution computed based on the counts of patterns in the tree of the training image. It uses the procedure getCPDF to compute the marginal conditional distributions. Like UpdateTree it never changes the shape of the tree as it only add contributions from patterns that can also be found in the training image.

   The inputs of the procedure is the same as of wrapUpdateTreeBoundary.

B.13 GrowTree

This procedure is called by SimNewImage. It is used to iteratively update the tree of an image when the latter is being perturbed. For each changed voxel value up to sN + 1 patterns may have changed and the tree needs to be updated for each of these changes.

   GrowTree is called each time a voxel value has been changed, and it uses of the procedures CenterCount, AddCount, SubtractCount and wrapUpdateTreeBoundary to update the tree. Out of the possibly sN+1 changed patterns, updating the tree with respect to the pattern of the changed voxel is relatively simple. However, it is done differently depending on whether the voxel is an inner voxel or not, as this makes it contribute differently to the tree.

   The changed voxel might be a neighbour of up to sN voxels, and it therefore might be a part of equally many other patterns. By changing the voxel value these patterns have changed too. The updating of these patterns is a bit tricky and depends on whether or not the changed voxel, as well as

the voxels of which it is a neighbour, are inner voxels or not. Patterns not centred in inner voxels are handled by the same procedure as when the tree was first constructed, namely wrapUpdateTreeBounday.

When pattern change it might happen that they go from not being a part of the tree to being part of the tree. The way the perturbation of images is done this will often be the case. It might also happen the opposite, namely that patterns used to be in the tree but are not any more. The number of counts in the frequency distribution of the image is for the same reason varying in the different iterations of the simulated annealing algorithm.

Variable	Description	Type
Z	The image after the voxel value has been changed.	3D integer array
zold	Old value of the changed voxel.	integer
Zex	Array used by getNewIt, should be updated according to the new patterns created by changing the voxel values.	3D logical array
i,j,k	Indices of the changed voxel.	integers
nodes	Array from the Nmask structure.	2D integer array
sV	Parameter, the images contain the $sV + 1$ categories $0, 1, \ldots, sV$ of voxel values.	integer
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer
Ttrain	Tree of patterns extracted from the training image.	tree
T	Tree of the image before the voxel was changed. The procedures updates it according to the new image Z.	tree

B.14 CenterCount

This is a recursive procedure called by GrowTree. The procedure is used to update the tree with respect to the pattern centred in the changed voxel when it is an inner voxel. Due to the structure of the tree, the tree is updated by following the (unchanged) voxel values of the neighbourhood voxels and updating the counts of the repl arrays of the corresponding nodes. Say the value of the voxel was changed from i to j then the repl arrays are updated by subtracting 1 count from the $i+1$th element and adding one count to the $j+1$th element.

This accounts for the updating for 1 out of the sN+1 patterns possibly affected as explained in the description of GrowTree.

Variable	Description	Type
T	Tree node to be updated	tree
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $\mathsf{sN}+1$ voxels.	integer
znew	New value of the changed voxel	integer
zold	Old value of the changed voxel	integer
Zvec	Holds the values of the voxels that are in the neighbourhood of the changed voxel.	1D integer array
exist	Indicate whether the pattern is found in the training image (exist = true) or not (exist =false).	logical

B.15 AddCount

This is a recursive procedure called by GrowTree. The procedure is used to add a count representing new patterns appearing when the image is perturbed. It is used when the voxel, which value was changed, was an inner voxel. This procedure performs the updating of the patterns for those of the sN neighbouring voxels, that are inner voxels.

It loops through those inner voxels, determines which patterns they are now centres of, and adds the count in the tree. Only one value has changed, namely the one belonging to the changed voxel, and the remaining $sN - 1$ values are unchanged. Therefore, if the changed voxel is the ith neighbour in the pattern, then the $i-$!1th first values of the pattern are unchanged and the tree should only be altered from level i and deeper.

Variable	Description	Type
T	Tree node to be updated	tree
sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of $sN + 1$ voxels.	integer
level	Depth level of the tree where the changed voxel will first have effect.	integer
Zvec	Holds the values of the neighbouring voxels in the changed pattern.	1D integer array
zcen	Value of the neighbouring voxel that is center in the changed pattern	integer
exist	Indicate whether the pattern is found in the training image (exist = true) or not (exist =false).	logical

B.16 SubtractCount

Like AddCount this is a recursive procedure called by GrowTree. Also this procedure is used to update the tree when the image is perturbed. It is used to subtract the count of the old pattern. It starts from the depth of change in voxel values and goes all the way to the bottom of the tree.

Variable	Description	Type
T	Tree node to be updated	tree

sN	Parameter, number of neighbours for an inner voxel resulting in patterns consisting of sN + 1 voxels.	integer
level	Depth level of the tree where the changed voxel will first have effect.	integer
Zvec	Holds the values of the neighbouring voxels in the changed pattern.	1D integer array
zcen	Value of the neighbouring voxel that is center in the changed pattern.	integer