Statistical Image Analysis

Computer Exercise 4: Statistical Image Models

Mats Kvarnström
Department of Mathematical Statistics
Chalmers University of Technology

January 2005

1 Introduction

In this exercise we are going to investigate some statistical image models. This will be done by simulating. First we deal with independent random processes which are usually used as a model of the noise in an image. Then, we turn our attention to dependency between the pixel values using the concept of Markov Random Fields. To illustrate this, we use the Ising model.

2 White noise

If the pixel values are random and uncorrelated they are said to be white noise. The (marginal) distribution of each pixel value may be of any kind; as long as the values are uncorrelated, it is white noise.

In this section we are going to simulate processes with independent pixel values (which implicate that they are uncorrelated, and thereby white noise). These kind of images are the easiest to simulate, simply because you simulate each pixel value in the same way, regardless of the values of the other pixels. Matlab has two useful commands for generating random variables. The first one, `rand`, generates uniformly distributed random variables between zero and one, and the second `randn` generates a normally (Gaussian) distributed random variable with expectation zero and variance 1. (It should be mentioned that Statistics Toolbox has commands for all common distributions. Type `help stats` for a list of the commands and functions in this toolbox.)

2.1 Simulation of Gaussian white noise

To simulate an image of size 64x64 consisting of white Gaussian noise (not what we usually would call an *image*, though) with mean $\mu = .5$ and standard deviation $\sigma = .2$, write:

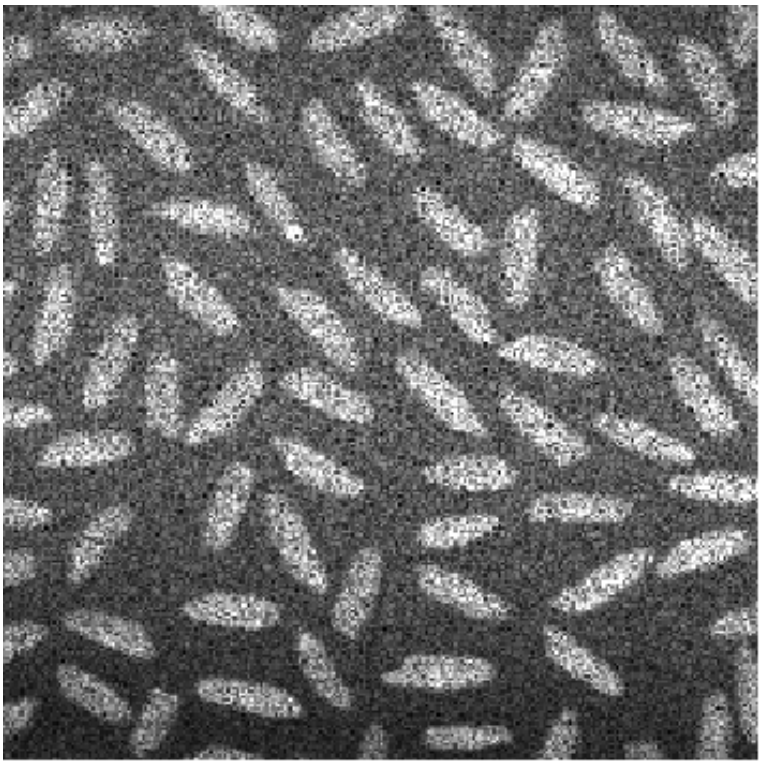

Figure 1: Multiplicative white Gaussian noise with $\sigma = 0.3$ corrupting the 'rice.png' image.

```
>>Ngauss=.2*randn(64) + .5;
```

Notice that `randn(64)` produces a *square* matrix of size 64x64 of random N(0,1)-distributed variables. If it is crucial to have pixel values only in the interval $[0, 1]$, you can for example use `find` to truncate:

```
>>Ngauss(find(Ngauss>1))=1;
```

Do analogously to truncate the pixel values less than zero.

The noise image itself is not really interesting. What we interested in, is when noise degrade or corrupt a real image, and how (and if) an image processing algorithm works even if we have a noisy image.

Load the image `'rice.png'` and corrupt the image by adding or multiplicate each pixel in the image by a Gaussian random variable. Adding noise can be done by the following:

```
>>I=imread('rice.png');
>>I=im2double(I);
>>Na=.15*randn(256);
>>I_add=I+Na;
>>figure(1),imshow(I_add)
```

Notice that `Na` is a 256x256 matrix of independent normal variables with mean zero and standard deviation .15. For the multiplicative noise we do:

```
>>Nm=.3*randn(256)+1;
>>I_mult=I.*Nm;
>>figure(2),imshow(I_mult)
```

You should try the above for different variances. Notice that multiplicative noise should have mean 1 if you want it to be unbiased.

Figure 1 shows one realization of the multiplicative noise above (since noise is random, it should be different every time you use `randn` or `rand`).

2.2 Impulsive noise

Now, try to distort the image by adding some 'pepper-and-salt' to it. By this, we mean that every pixel have chance p of becoming either black or white (with equal probability), independent of each other. This can be done by a uniformly distributed random number U in the range $[0, 1]$ for each pixel and set the corresponding pixel value to black if $U < p/2$ and to white if $U > 1 - p/2$. In Matlab code this might look like:

```
>>U=rand(256); %256x256 independent U[0,1] variables
>>p=.1;
>>I_ps=I;
>>I_ps(find(U<p/2))=0;
>>I_ps(find(U>1-p/2))=1;
>>figure(3),imshow(I_ps)
```

Try this for different p-values and look at the resulting, corrupted image.

2.3 Filtering

Now you may ask yourself the following questions: How can these different kind of noise corrupted images be restored and does the noise degrade the performance of image analysis procedures?

To get a feeling for this, you should apply the filters from Part I in Computer Exercise 2 on the three noise-corrupted images. Try also the median filter (implemented in Matlab as `medfilt2`); for which of the three different kinds of noise does this filter restore the image particularly well?

In Figure 2, the difference of applying Prewitt's edge detection filter (see Computer Exercise 2) to the original versus the noise-corrupted image, is illustrated.

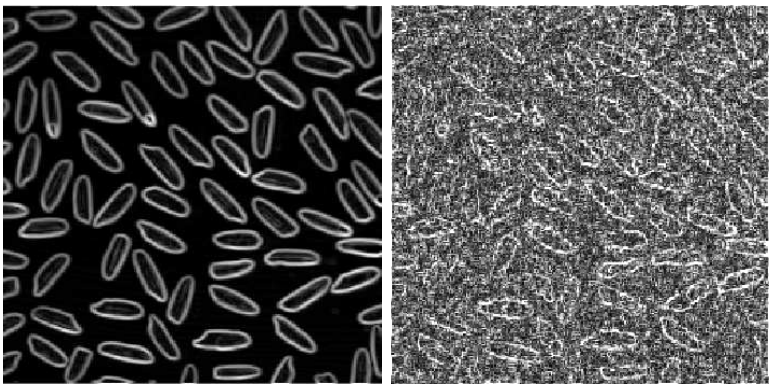

Figure 2: The result after applying Prewitt's filter to the original 'rice.png' image (left), and the noise-corrupted image in Figure 1 (right).

3 Simulation of Markov Random Fields

An appealing way of modelling the dependence between pixel values at different sites is by a Markov Random Field. Let $X = (X_s, s \in S)$ be a set of random variables taking values in the set V. We say that X is a Markov Random Field with respect to the neighbourhood system $(N_s, s \in S)$ if

$$\Pr(X_s = x \,|X_t, t \neq s) = \Pr(X_s = x \,|X_t, t \in N_s),\ x \in V,\ s \in S \qquad (1)$$

where S denotes the set of pixel locations (sites).

In words (and somewhat loosely), what equation (1) says, is simply that the pixel value at $s \in S$, given all the other pixel values, only depends on the values of the pixels neighbouring (with respect to the neighbourhood system N_s) to s. See Chapter 3 in the Lecture Notes [3].

3.1 The Ising Model

To be concrete, let S be the vertices in the square lattice of size N and let the neighbourhood system be given by, for $1 < i, j < N$ (i.e. for non-border sites)

$$N_s = \{(i-1, j), (i+1, j), (i, j-1), (i, j+1)\} \qquad (2)$$

and let X_s take values in $V = \{-1, 1\}$. The Ising Model says that

$$\Pr(X_s = +1 | X_t, t \in N_s) = \frac{\exp(2\beta(X_s^+ - X_s^-))}{1 + \exp(2\beta(X_s^+ - X_s^-))} \qquad (3)$$

where X_s^+ and X_s^- denotes the number of neighbours of s taking the value +1 and -1, respectively. Furthermore, β is a parameter called the inverse temperature.

3.2 The Gibbs Sampler

A widely used algorithm for simulation of Markov Random Fields is the Gibbs Sampler. It is a so called Markov Chain Monte Carlo (MCMC) method and it looks like this

1. Choose site $s \in S$ either randomly (uniformly over S) or deterministically (taking each site row-wise from left to right).
2. Examine the neighbourhood to s, N_s.
3. Sample the (new) pixel value of according to the conditional distribution given the neighbours $(X_t, t \in N_s)$ to s. Notice that it does not matter what the value of the site is, only the values of its neighbours.
4. Choose the next site s according to the chosen rule, and go to 2.

The above should be repeated until we reach the stationary distribution of the Markov Chain. When this happens is in general not known. Often there only exists guidelines for how long you should run your Markov Chain. In the examples in this exercise we are talking about a maximum of maybe a hundred sweeps (one sweep = one visit per site in a row-wise scan from left to right).

For a thorough exposition of Markov Chains and MCMC, see Häggström [2], where also the concept of *perfect* or *exact* sampling is dealt with.

3.3 Implementation and simulation

The idea now is to simulate a Markov Random Field or, more specific, the Ising Model. Choose S to be the vertices of a square lattice of size 64 (i.e. a square matrix of size 64x64) and let $V = \{-1, 1\}$ be represented by $V = \{0, 1\}$. We use periodic boundaries by which we mean that the right boundary is connected to the left boundary and the upper to the lower and vice versa.

It should be quite straightforward to implement the Gibbs Sampler. We basically need two functions:

- One that looks up the pixel values of the neighbours to a given site. This we already have from CE2, except that you have to modify it so it can handle the periodic boundaries.
- A function that handles the update procedure of a pixel value. Calculate the probability $p = \Pr(X_s = 1 | X_t, t \in N_s)$ given by (3) and draw a random number U in $[0, 1]$. If $U \leq p$ set the pixel value to 1, otherwise 0.

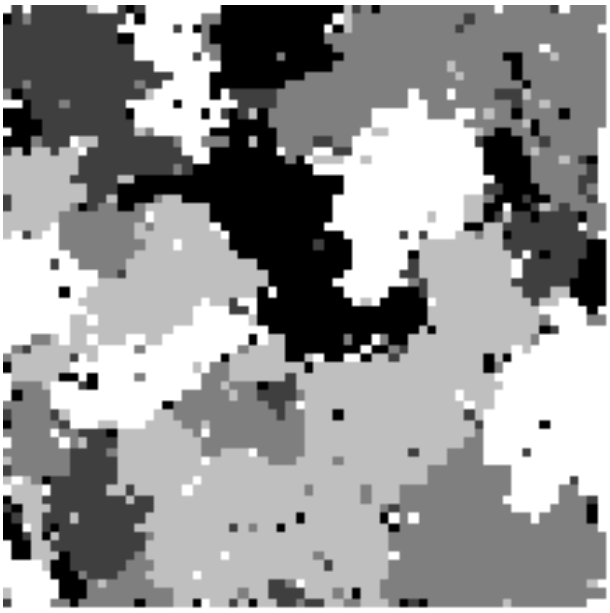

Figure 3: A sample from an MRF using the Potts model with 5 states (the Ising model is a Potts model with 2 states).

The starting image can be any binary image. The Markov Chain converges regardless of the initial configuration. The rate of convergence may depend on this, though. So, choose a purely random start configuration, i.e. each pixel having a probability of a half of being either black of white, independently of each other. (However, you are encouraged to try other starting configurations.)

See the Appendix for the outline of a main program to the Gibbs Sampler.

3.4 Relevance to image analysis

You may ask why we do simulate the Ising model and what does that have to do with Image Analysis? The answer is that the Ising model is just an example of a Markov Random Field (probably the simplest, non-trivial model), which in general could be far more advanced. If we expand the set V to the gray scale and alter our updating rule to a rule with a larger neighbourhood system and that, at the same time, considers possible edges in the image based on prior knowledge of what images 'usually looks like'?

To illustrate the first way to expand the model, first look at Figures 3 to 5. The image in Figure 3 was created using the Potts model with 5 states (one can say that the Potts model is a generalization of the Ising model to more than two states) in the Gibbs sampler. We think of this image as the original image. This image is corrupted by adding white Gaussian noise

Figure 4: The image from Figure 3 with added noise.

and rounding so that the pixels in the corrupted image take values in the original 5 states, resulting in the image in Figure 4.

Now, say that we want to recover the original image (in Figure 3) given the image in Figure 4. Since we know the underlying statistical model of the original image (the Potts model) we should be able to use this information for the recovery. Here is a possible way of doing this: we run the Gibbs Sampler once again, with the corrupted image as a start image, but this time with a lower temperature (higher β) and as we iterate, slowly decreasing it. This will make the model less tolerant to spurious changes of intensities, the idea being that the noise should be suppressed. This procedure is called *simulated annealing* (see [2]). By comparing Figure 5 and 3, we see the we recovered the original image fairly well.

So, if we could knew the model that 'created' a real image, noise reduction would be easy; just plug the distorted image into a Gibbs Sampler with this model and iterate. The problem here, is naturally the complexity of this (hypothetical) model. It is quite fascinating though and with the increasing power and speed of modern computers the barrier of the immense complexity of this task is getting smaller. The ground breaking article regarding this approach was written by Stuart and Donald Geman [1] in 1984, and a lot of research in this area has been done since then. For a more recent article, see the one by Song Chun Zhu and David Mumford [4].

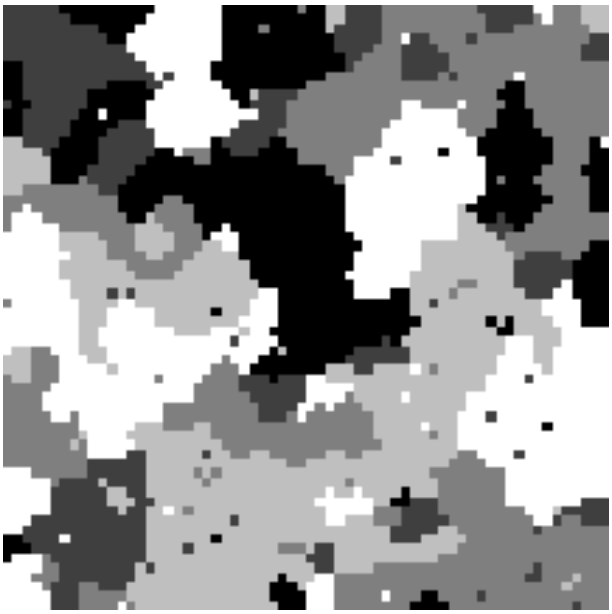

Figure 5: The resulting image after 10 iterations of simulated annealing.

References

[1] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 721–741, 1984.

[2] Olle Häggström. *Finite Markov Chains and Algorithmic Applications.* Cambridge University Press, 2002.

[3] Mats Rudemo. *Image Analysis and Spatial Statistics.* Dept. of Mathematical Statistics, Chalmers University of Technology, 2003.

[4] Song Chun Zhu and David Mumford. Prior learning and gibbs reaction-diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1236–1250, 1997.

Appendix

```
%Outline of a main program for simulation of the Ising model
%
%Mats K 010208

%The parameter called the inverse temperature:
beta=.5;
%The size of the square image:
```

```
N=64;
%Start image:
I=rand(N)<.5;
%K=the total number of iterations:
K=100;

for s=1:K
   %The two loops below constitute a 'sweep'
   for k=1:N
       for l=1:N

          %First: look up the neighbours to I(k,l)
          %Don't forget to take care of the periodic
          %boundaries:
          n=neighbours(I,[l,k]);

          %Now, sample from the conditional distribution:
          I(k,l)=update_ising(n,beta);

       end
   end

   %Show the result from this sweep and pause for .1 seconds
   %This step should be erased when you see that it works
   %as it should.

   imshow(I),pause(.1)
end
imshow(I)
```