Computer problems 2: Markov Chain Monte Carlo methods

Getting Started

If you work on the NADA computer system: Before entering Matlab, copy the file `efronmain.m` to your own working directory:

```
cp /info/fysikkurser/matstat/efronmain.m efronmain.m
matlab
```

Here `efronmain.m` creates paths to all functions you need for this assignment. It should be processed as soon as you enter Matlab. The software is written by Finn Lindgren and the assignments on MCMC are constructed by Ola Hössjer and Finn Lindgren.

You can also get the files from the course home page.

1 Simulating the hard core model

$\chi = I \times J$-lattice with 0:s and 1:s and let the state space be
$E = \{x \in \chi$ two neighbours are both not 1 1$\}$.
The allowed configurations are therefore $I \times J$-matrices of 0:s and 1:s but where there can not be two 1:s next to each other vertically och side-ways. If we take $J = I = 10$ we have a lattice with 100 points and we realize that it is not even easy to calculate the number of "allowed" configurations.

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

is an "allowed" matrix

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

is not "allowed". We want to choose an "allowed" matrix with the same probability for each of the "allowed" matrices of a fixed size.

Let π be a distribution on χ which is 0 outside E, e.g. $\pi(x)$=constant for $x \in E$ and 0 otherwise (uniform distribution on E). Notice that we in fact do know how large E is!

As a proposal distribution (i.e. $q(x, y)$) we can take:

Choose node (i, j) at random with probability $1/IJ$. Choose the value in the node as 0 or 1 with probabiliy 1/2 each.

Possibly we get $y \notin E$ i.e. that the proposal gives a configuration which is not allowed, but then $\pi(y) = 0$ and therefore $\alpha(x, y) = 0$, i.e. such a proposal is ignored and the chain stays in the same position.

If $y \in E$ (i.e. an "allowed" configuration) we accept the proposal since

$$\alpha(x, y) = \min\left(1, \frac{\pi(y)q(y, x)}{\pi(x)q(x, y)}\right) = 1$$

since $q(x, y) = q(y, x)$ and $\pi(x) = \pi(y)$.

The chain is obviously irreducible and aperiodic, i. e. it is ergodic. We see this is true since given two configurations x and y we can start by removing all 1:s in x and then add the 1:s in y - this shows the irreducability. Obviously we have a positive probability to stay in the same configuration which shows aperiodicity. Since the state space is finite it is therefore ergodic.

1) We are interested in the number of 1:s in such a matrix selected at random. This number is of course random.
Calculate the exact distribution for the number of 1:s for 2×3-matrices by enumeration of all possible cases.

2) We now study 10×10-matrices. Use simulation as above to estimate the expected number of 1:s in such a randomly selected matrix and it variance.

Let further
U=the number of 1:s in the row with the maximum number of 1:s and
V=the number of 1:s in the row with the minimum number of 1:s and
$X = U - V$, i.e. the difference in the number of 1:s between the row with maximum and the row with minimum number of 1:s. What values are possible for this random variable X? Determine (approximately) the probability distribution of X by MCMC-simulation. Calculate also (approximately) the mean value and variance in this distribution.
Hint: Put a frame of 0:s around your 10×10-matrix in order to get rid of boundary effect.

Note that the procedures `max` and `sum` in Matlab work column-wise if they are applied to matrices.

2 Simulating a Bivariate Normal Distribution

2.1 Independent Components

You will use `snorm` (or possibly `Esnorm`) to generate samples from a bivariate normal distribution

$$N_2\left((m_1, m_2)^T, \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}\right) = N_2(\boldsymbol{m}, \boldsymbol{V}),$$

with either of the Choleski, Gibbs or Metropolis-Hastings random walk algorithms. Take a look at the function first:

```
>> help snorm
```

A bivariate normal distribution has 5 parameters, and they are gathered in `th`. Simulate 200 observations from a standard normal bivariate distribution using the Choleski method and plot the result:

```
>> th = [0 0 1 1 0]
>> [X ml cl] = snorm(th,200,'Ch')
>> plot(X(:,1),X(:,2),'.')
```

The ouputs `ml` and `cl` are loss functions for estimation of the mean and covariance matrix:

$$\begin{aligned}\texttt{ml} &= \|\boldsymbol{V}^{-1/2}(\text{mean}(X) - \boldsymbol{m})\| \\ \texttt{cl} &= \|\boldsymbol{V}^{-1/2}(\text{cov}(X) - \boldsymbol{V})\boldsymbol{V}^{-1/2}\|,\end{aligned}$$

where mean and cov are MATLAB functions operating on the output sample X and $\|\cdot\|$ is the Euclidean norm in two and four dimensions respectively. By this we mean

$$\|(v_1, v_2)^T - (u_1, u_2)^T\| = \sqrt{(v_1 - u_1)^2 + (v_2 - u_2)^2}$$

and

$$\left\| \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} - \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \right\| = \sqrt{\sum\sum_{i,j=1}^{2} (a_{ij} - b_{ij})^2}.$$

Now `ml` and `cl` measure the quality of the generated random vectors in two different ways: How much does mean(X) (the sample mean) and cov(X) (the sample covariance matrix) differ from the true expected value $\boldsymbol{m}$ and covariance matrix $\boldsymbol{V}$?

Since the Choleski method generates perfect i.i.d. random vectors we can derive the distributions of `ml` and `cl` (although the second one is more difficult).

1. Assume first a parameter vector `th = [0 0 1 1 0]`. Using the fact that mean$(X) \in N_2((0,0)^T, \boldsymbol{I}_2/n)$, it can be shown that `ml` is Rayleigh distributed with density function

   $$f(x) = nxe^{-nx^2/2}, \quad x > 0.$$

   Compute the expected value of `ml`. (Hint: Write the integrand as $xf(x) = x \cdot nxe^{-nx^2/2}$ and use integration by parts. Further, the formula $\int_0^\infty e^{-t^2/2}dt = \sqrt{\pi/2}$ will be useful.)

2. Prove that the distribution of `ml` is invariant (does not depend on `th`). This means that the conclusion in 1 holds for any bivariate normal distribution. (Hint: Use the representation $X(k,:) = \boldsymbol{m} + \boldsymbol{V}^{1/2}\boldsymbol{\varepsilon}_k$ where $\boldsymbol{\varepsilon}_1, \ldots, \boldsymbol{\varepsilon}_n$ are i.i.d. $N_2((0,0)^T, \boldsymbol{I}_2)$ random vectors. This means mean$(X) = \boldsymbol{m} + \boldsymbol{V}^{1/2}$mean$(\boldsymbol{\varepsilon})$, where mean$(\boldsymbol{\varepsilon})$ is the mean of all $\boldsymbol{\varepsilon}_k$ vectors and has the same distributon as mean(X) in 1.)

If we plot the two loss functions versus sample size, the curves will be very noisy. It is better to average over several realizations, using the function `snorm2` (or possibly `Esnorm2`). When `iter`, the number of iterations, is fairly large, we can plot estimates of E(`ml`) and E(`cl`) versus sample size as follows: (If your machine is fast, try a larger value of `iter`.)

```
>> help snorm2
>> n = [50 100 200 500 1000]'
>> iter = 5;
>> [ml(1) cl(1)] = snorm2(iter,th,n(1),'Ch');
>> [ml(2) cl(2)] = snorm2(iter,th,n(2),'Ch');
>> [ml(3) cl(3)] = snorm2(iter,th,n(3),'Ch');
>> [ml(4) cl(4)] = snorm2(iter,th,n(4),'Ch');
>> [ml(5) cl(5)] = snorm2(iter,th,n(5),'Ch');
>> figure(1)
>> plot(n,ml','-')
>> hold on
>> figure(2)
>> plot(n,cl','-')
>> hold on
```

If you want, you can include the exact value of $E(\texttt{ml})$ (computed in 1.) in Figure 1.

Now turn to the Gibbs sampler. If only three input arguments are given, the Markov chain starts at $(0,0)^T$ and no iterates are discarded in the beginning (`n_init = 0`).

```
>> [ml(1) cl(1)] = snorm2(iter,th,n(1),'Gi');
>> [ml(2) cl(2)] = snorm2(iter,th,n(2),'Gi');
>> [ml(3) cl(3)] = snorm2(iter,th,n(3),'Gi');
>> [ml(4) cl(4)] = snorm2(iter,th,n(4),'Gi');
>> [ml(5) cl(5)] = snorm2(iter,th,n(5),'Gi');
>> figure(1)
>> plot(n,ml','-.')
>> hold off
>> figure(2)
>> plot(n,cl','-.')
>> hold off
```

Is there any systematic difference between the two methods? If not, can you explain it?

As the last method, we consider the Metropolis-Hastings algorithm with a random walk kernel

$$q(x,y) = q_1(y-x); \quad q_1 \sim \mathrm{N}_2\left((0,0)^T, s^2\boldsymbol{I}_2\right).$$

The standard deviation s determines the size of the jumps. There is also a new output parameter `acc` that determines the proportion of accepted moves. Generate samples of length $n = 500$ and varying s. Choose `iter` so that the computation time is feasible.

```
>> s = [0.2 0.4 0.7 1 2 4]'
>> [ml(1) cl(1) acc(1)] = snorm2(iter,th,500,'MH',s(1));
>> [ml(2) cl(2) acc(2)] = snorm2(iter,th,500,'MH',s(2));
>> [ml(3) cl(3) acc(3)] = snorm2(iter,th,500,'MH',s(3));
>> [ml(4) cl(4) acc(4)] = snorm2(iter,th,500,'MH',s(4));
>> [ml(5) cl(5) acc(5)] = snorm2(iter,th,500,'MH',s(5));
>> [ml(6) cl(6) acc(6)] = snorm2(iter,th,500,'MH',s(6));
>> plot(s,ml','-.',s,cl',':',s,acc','-')
```

How does s affect the acceptance rate and the loss functions?

2.2 High correlation

Now redo everything for a distribution with high correlation 0.95:

```
>> th = [0 0 1 1 0.95]
>> [X ml cl] = snorm(th,200,'Ch')
>> plot(X(:,1),X(:,2),'.')
```

Repeat all the other steps above for the Choleski, Gibbs and MH algorithms. Some questions:

- Are the loss functions for the Choleski method affected?
- Is the performance of the Gibbs sampler still comparable to the Choleski method?
- How does correlation affect acceptance rate and performance of the MH algorithm?

Before moving to the next section, type

```
>> clear
```

3 Simulating a Two Component Mixture (Multimodality)

We will now simulate the two component mixture

$$0.5\mathrm{N}_2\left((0,0)^T, \boldsymbol{I}_2\right) + 0.5\mathrm{N}_2\left((7,7)^T, \boldsymbol{I}_2\right).$$

This means the with equal probabilities 0.5 the first and second normal components are chosen respectively. The two distributions are defined through

```
>> th1 = [0 0 1 1 0]
>> th2 = [7 7 1 1 0]
```

Start generating and plotting 500 observations with the Choleski method. (The mixture probability $p = 0.5$ is specified as the last input argument.)

```
>> [X ml cl acc ph] = snorm(th1,500,'Ch',[],[],[],th2,0.5)
>> plot(X(:,1),X(:,2),'.')
```

The output `ph` is the relative frequency of visits to the second component. It can be interpreted as an estimator of the mixture probability p. Which is the distribution of `ph`?

Now try the Gibbs sampler (with `n_init` $= 0$ and starting point $(0,0)^T$).

```
>> [X ml cl acc ph] = snorm(th1,500,'Gi',[],[0 0],0,th2,0.5)
>> plot(X(:,1),X(:,2),'.')
```

Is the second mixture component ever visited? Why? Can you think of any modification of the Gibbs algorithm with better performance for this distribution?

Try again, but choose the second mean closer to the origin:

```
>> th3 = [4 4 1 1 0]
>> [X ml cl acc ph] = snorm(th1,500,'Gi',[],[0 0],0,th3,0.5)
>> plot(X(:,1),X(:,2),'.')
```

Did you notice any difference?

As in the previous section, we investigate how the spread parameter s affects the MH-algorithm. The value of `iter` may have to be decreased if the computation times are too high.

```
>> s = [0.5 1 3 5 7 9]'
>> x0 = [0 0];
>> [ml(1) cl(1) acc(1) ph(1)]=snorm2(5,th1,500,'MH',s(1),x0,0,th2,0.5)
>> [ml(2) cl(2) acc(2) ph(2)]=snorm2(5,th1,500,'MH',s(2),x0,0,th2,0.5)
>> [ml(3) cl(3) acc(3) ph(3)]=snorm2(5,th1,500,'MH',s(3),x0,0,th2,0.5)
>> [ml(4) cl(4) acc(4) ph(4)]=snorm2(5,th1,500,'MH',s(4),x0,0,th2,0.5)
>> [ml(5) cl(5) acc(5) ph(5)]=snorm2(5,th1,500,'MH',s(5),x0,0,th2,0.5)
>> [ml(6) cl(6) acc(6) ph(6)]=snorm2(5,th1,500,'MH',s(6),x0,0,th2,0.5)
>> figure(1)
>> plot(s,ml','-.',s,cl',':')
>> figure(2)
>> plot(s,acc','-',s,ph','--')
```

Looking at the plots, notice in particular how `ph` depends on s.

4 Simulating the Ising Model

The Ising model gives a simple description of a magnet. Suppose we have an $m \times n$ lattice

$$S = \{(i,j);\ 1 \le i \le m,\ 1 \le j \le n\}$$

and a vector

$$\boldsymbol{X} = (X_{(i,j)};\ (i,j) \in S).$$

Each component $X_{(i,j)} \in \{-1, 1\}$ represents the magnetic spin of an atom. We assume that $\boldsymbol{X}$ is an observation of a Gibbs distribution π on $\{-1,1\}^S$:

$$\begin{aligned} \pi(\boldsymbol{X}) &= \exp(-\tilde{\beta} E(\boldsymbol{X}))/Z, \\ E(\boldsymbol{X}) &= -\textstyle\sum_{(i,j)\in S} \sum_{(i',j')\in N(i,j)} x_{(i',j')} x_{(i,j)}. \end{aligned}$$

Here E is the energy function, $\tilde{\beta}$ a parameter describing the strength and type of magnetic interaction and Z a normalization constant that makes π a probability distribution function. $N(i,j)$ is the array of neighbours to (i,j). For instance, a neighbour structure

$$N = \begin{matrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{matrix}$$

gives four neighbours of (i,j) (the neighbours are marked with 1 and (i,j) is the central 0). Since π has 2^{mn} possible outcomes, it is extremely time consuming to simulate π directly.

Fortunately, the conditional distibution of the components are easy to update, which makes the Gibbs sampling procedure appropriate. Verify that

$$\pi\left(X_{(i,j)}=1|\boldsymbol{X}_{-(i,j)}\right)=\frac{\exp\left(2\tilde{\beta}(\mathrm{NN}_1(i,j)-\mathrm{NN}_{-1}(i,j))\right)}{\exp\left(2\tilde{\beta}(\mathrm{NN}_1(i,j)-\mathrm{NN}_{-1}(i,j))\right)+1}, \tag{1}$$

with $\mathrm{NN}_\nu(i,j)$ the number of neighbours of (i,j) with value ν. The vector $\boldsymbol{X}_{-(i,j)}$ contains all components in $\boldsymbol{X}$ except $X_{(i,j)}$.

We will now use

```
function M = film(iter,beta,N,omega0)
```

to generate Gibbs samples from π. The program codes -1 to 0, so all arrays are binary from now on. Define the (random) starting configuration and neighbour structure:

```
>> omega0 = rand(20,20)<0.5,
>> N = [0 1 0; 1 0 1; 0 1 0]
```

The program uses

$$\tilde{\beta}=\frac{\beta}{2\mathrm{NN}(i,j)} \tag{2}$$

when updating the components. Here $\mathrm{NN}(i,j)=\mathrm{NN}_{-1}(i,j)+\mathrm{NN}_1(i,j)$ is the number of neighbours to (i,j) and β is held fixed. The formula compensates for NN being smaller at egdes and corners, and makes the interaction stronger at such locations. It corresponds to "there is nothing outside the lattice", whereas a constant $\tilde{\beta}$ would force us to put a boundary condition outside S.

Define β and number of iterations:

```
>> beta = 1
>> iter = 30
```

One iteration means that all mn components are updated rowwise with the correct conditional probabilities. Run a sequence of `iter` simulations: The function `film` generates the movie, puts the result into `M` and then `movie` runs several replays back and forth. A third optional parameter of `movie` controls the speed of the replay. Warning: Choose a rather small graphical window, otherwise you may run out of memory! All displays may be interrupted with `ctrl C`.

```
>> M = film(iter,beta,N,omega0);
>> help movie
>> movie(M,-100)
```

Now redo everything for other `beta` and neighbour structures, e.g.

```
>> N = [ 0 1 1; 0 0 0; 1 1 0].
```

For stability reasons, it is important that `N` is symmetric w.r.t. the origin. Larger `beta` may require more iterations before convergence. Some questions:

- How does `beta` influence the configurations?
- How does `N` influence the configurations?
- Can you find any analogy between a large `beta` (e.g. 10) and the bimodal example of Section 2.

5 Bayesian statistics

5.1 Binomial distribution

Let X be $\mathrm{Bin}(n, \theta)$ where θ has a discrete apriori distribution $p_\Theta(\theta)$, $0 < \theta < 1$ defined on a dense set of values. Therefore the likelihood function

$$L(\theta, x) = \binom{n}{x} \theta^x (1-\theta)^{n-x} = p_{X|\Theta=\theta}(x)$$

Let e.g. $n = 10$, $x = 2$ and $n = 100$, $x = 20$ and calculate with Matlab the aposteriori distribution numerically for $\Theta|X = x$. Use both a uniform apriori distribution and some non-uniform distribution. Let e.g. the lattice be defined by $p = 0.001 : 0.001 : 0.999$ i.e. the numbers 0.001 until 0.999 in steps of 0.001.

5.2 Exponential distribution

Generate 10 observations $x_1, x_2, \cdots, x_{10}$ which are exponentially distributed with mean value 5. Select some different apriori distributions (discrete ones mimicing continuous distributions) for the expected value θ in the exponential distribution and calculate the aposteriori distribution of θ given the observations. Take e.g. a uniform distribution $U(0, 30)$ and some other distribution as apriori distribution.

6 Problems on Simulated annealing

Use simulated annealing to solve the "over determined" binary system of equation $Ax = b$ where b is a binary $n \times 1$-vector (consisting of 0:s and 1:s) and A is a binary $n \times m$-matrix. The solution x is also a binary $m \times 1$-vector.

The matrix multiplication Ax is performed modulo 2, i.e. the result is itself a binary vector.

To find a "solution" $\widehat{x}$ we should choose a binary x such that Ax coincides with b in as many components as possible, i.e. $\widehat{x}$ minimizes

$$|Ax - b|, \text{ where } |\cdot| \text{ is the summation of the absolute values.}$$

Simulated annealing for this problem can be performed by using the function *anneal* which can be downloaded from the home page.

We have to generate a b-vector and an A-matrix and that can e.g. be done using the code

$$b = rand(n, 1) < 0.5; \quad A = rand(n, m) < delta;$$

where $n > m$, i.e. the system is over determined.

Suitable values can be $n = 100$ and $m = 50$ together with $delta = 0.1$. The latter parameter *delta* is the proportion of 1:s in the A-matrix and a large value makes the minimizing problem too difficult since $|Ax - b|$ then is quite discontinuous.

I have produced some A-matrices and b-vectors which can be found in $an4.mat$ and $an4ny.mat$ in the directory

$$http://www.math.kth.se/matstat/gru/5b1555/mfiler/$$

where they can be downloaded. The can then be loaded into Matlab using the code *load filnamn*.

The command *load* $an4$ gives you a b-vector and an A-matrix with $= 100$ and $m = 50$ and $delta = 0.1$ where I have found the minimal value $\underline{16}$. I do not know if this is the true minimum but the problem is too big (there are $2^{50} \approx 1.13 \cdot 10^{15}$ possible x-vectors) to solve the problem exactly (as far as I know).

By loading *an4ny* we get a b-vector of size 50×1 and an A-matrix of size 50×25 and hence there are $2^{25} \approx 33.000.000$ possible x-vectors. (I have investigated this problem and there is a unique solution given the value 6, i.e. with this special vector $\widehat{x}$ all equations except 6 match.

Use either the "prepared" problems or generate youselt an A-matrix and a b-vector and run simulated annealing using *anneal* - view the instructions using *help anneal*!

7 Using the package BUGS

For UNIX:

Arrange first so you can run BUGS by

1) either let the directory $/info/fysikkurser/matstat/bugs/$ be part of your path

2) or copy the executable files in that directory to a directory in your path (a suitable place is */bin* where program files are usually stored). The files are

$$bugs,\ bugs05.sparc,\ \text{ and } bugs06.sparc$$

In addition there is a file *backbugs* which makes it possible to run *BUGS* in the background.

Download the program file *pump.bug*, the indata file *pump.dat* and the initialization file *pump.in* from the *BUGS*-filedirectory on the course home page.
$http://www.math.kth.se/matstat/gru/5b1555/BUGS/$.
The files are also stored in the directory

$$/info/fysikkurser/matstat/bugs/$$

so they can be copied from there. (There are also files $dyes.bug$ and $dyes2.bug$ which are examples of onesided and twosided analysis of variance.)

View the files and try to understand the syntax. Remember that *BUGS* updates the different parameters using Gibbs sampling, i.e. it updates them successively with the conditional distribution for the current parameter given all the rest. *BUGS* in principle demands that these are conjugated, log-concave or discrete - otherwise *BUGS* will complain.

pump.bug describes the example on 10 pumps run for differents times and the number of failures has been noted. The model is that the number of failures of pump nr i (which has been run the time t_i) is Po(λt_i). As apriori distribution for λ we have $\Gamma(\alpha, \beta)$ where α is Exp(1) and β is $\Gamma(0.1, 1)$.

A manual for *BUGS* can be found on
$http : //www.mrc - bsu.cam.ac.uk/bugs/$
where you click on *Documentation* and then on *Bugs manual v*0.5

A set of examples can also be found. By downloading the file *exdir*05*.tar* from

$$http : //www.math.kth.se/matstat/5b1555/BUGS/$$

and unpacking using the command

$$tar\ xvpf\ exdir05.tar$$

a whole directory structure (in the current directory) containing examples. This can also be found in $/info/fysikkurser/matstat/bugs/$.

For DOS you download *exdir*05*.exe* and unpack it with the command *exdir*05 -o -d. These examples can worthwhile to study.

Start *BUGS* with the command *bugs*. You exit with the command *q*().

Compile the program *pump.bug* with the command
compile(“*pump.bug*″).
Let then the chain be updated in (e.g)100 steps with the command *update*(100) so that the dependence on the initialization values is decreased.

Using the commands *monitor*(*alpha*) and *monitor*(*beta*) you indicate that you want output for the variables *alpha* och *beta*

The command *update*(1000) indicates that you want 1000 observations from the aposterio distribution for (α, β). When you exit *BUGS* with *q*() the file *bugs.out* is created which contains these observations. In addition you get a column with numbering of the observations. First you get the α-observations and then the β-observations.

Load these into Matlab with *load bugs.out*. Remember how the file *bugs.out* is organized. Information on this can be found in the file *bugs.ind* which *BUGS* generates.

Plot the distributions of α and β. A good way of illustrating is to generate the predictive distribution for λ (i.e. for a new pump) by generating a $\Gamma(\alpha, \beta)$-observation for each (α, β)-observation by using the Matlab-procedure *gamrnd*. Note that in Matlab the second parameter in the Γ-distribution is the scale parameter (i.e.$1/\beta$).