Övningsuppgift 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 assignement. 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

Låt $\chi = I \times J$-gitter med 0:or och 1:or. och låt tillståndsrummet vara
$E = \{x \in \chi$ två grannar inte båda $1\}$.
De tillåtna konfigurationerna är alltså $I \times J$-matriser av 0:or och 1:or men där det inte kan stå två 1:or intill varandra i sidled eller höjdled. Om vi tar $J = I = 10$ har vi ett gitter med 100 punkter och man kan inse att det inte ens är en lätt uppgift att beräkna storleken av E, dvs att bestämma hur många "tillåtna" konfigurationer det finns!

spot

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

är en "tillåten" matris och

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

är "otillåten". Vi vill välja en "tillåten" matris på måfå med lika sannolikhet för alla "tillåtna" matriser av en viss storlek.

Låt π vara en fördelning på χ som är 0 utanför E, t ex $\pi(x)$=konstant för $x \in E$ och 0 för övrigt (likformig fördelning på E). Notera att vi faktiskt inte vet hur stort E är!

Som förslagsfördelning (dvs $q(x, y)$) kan vi ta:

Välj nod (i, j) på måfå med sannolikhet $1/IJ$. Välj värde i noden som 0 eller 1 med sannolikhet $1/2$ vardera.

Eventuellt inträffar att $y \notin E$ dvs att förslaget innebär en "otillåten" konfiguration, men då är $\pi(y) = 0$ och alltså $\alpha(x, y) = 0$, dvs sådana förslag ignoreras och kedjan ligger kvar där den var.

Om $y \in E$ (dvs en tillåten konfiguration) accepteras förslaget ty

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

eftersom $q(x, y) = q(y, x)$ och $\pi(x) = \pi(y)$.

Kedjan är uppenbarligen irreducibel och aperiodisk, dvs ergodisk! Man kan inse detta genom att givet två konfigurationer x och y så kan vi börja med att plocka bort alla 1:or i x och sen sätta dit 1:orna som ingår i y - detta visar irreducibiliteten. Uppenbarligen har vi positiv sannolikhet att ligga kvar i t o m varje tillstånd vilket visar aperiodiciteten. Eftersom kedjan är ändlig är den ergodisk.

1) Vi är här speciellt intresserade av antalet 1:or i en sådan på måfå vald matris. Detta antal är naturligtvis en stokastisk variabel.
Bestäm den exakta fördelningen för antalet 1:or för 2×3-matriser genom ren uppräkning av alla tänkbara fall.

2) Vi studerar nu 10×10-matriser. Bestäm med simulering enligt ovan förväntat antal 1:or i en sådan på måfå vald matris samt variansen för detta antal.

Låt vidare
U=antalet 1:or i raden med flest 1:or samt
V=antalet 1:or i raden med minst antal 1:or samt
$X = U - V$, dvs skillnaden mellan raden med flest och med minst antal 1:or. Vilka värden är möjliga för denna stokastiska variabel X? Bestäm (approximativt) sannolikhetsfördelningen för X genom MCMC-simulering. Beräkna också (approximativt) medelvärde och varians i denna fördelning.
Tips: Lägg en "ram" av 0:or kring din 10×10-matris så slipper Du ta hänsyn till randeffekter.

Notera att procedurerna `max` och `sum` i Matlab funkar kolonn-vis om de har matrisvärda argument.

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

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

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

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

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 $\{-1,1\}^S$:

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

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 Bayesiansk statistik

5.1 Binomialfördelning

Låt X vara $\text{Bin}(n, \theta)$ där θ har en (diskret) aprioriförde lning $p_\Theta(\theta)$, $0 < \theta < 1$ definerad på en (förhållandevis tät) uppsättning värden. Alltså är likelihoodfunktionen

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

Låt t ex $n = 10$, $x = 2$ och $n = 100$, $x = 20$ och beräkna med Matlab aposterioriförde lningen numeriskt för $\Theta|X = x$. Använd dels en likformig apriori-fördelning och någon icke-likformig aprioriförde lning. Låt t ex gittret vara definierat av $p = 0.001 : 0.001 : 0.999$ dvs talen 0.001 t o m 0.999 i steg om 0.001.

5.2 Exponentialfördelning

Generera 10 st Exp(5)-fördelade observationer $x_1, x_2, \cdots, x_{10}$. Tag några olika aprioriförde l-ningar (diskreta som "härmar" kontinuerliga) för väntevärdet θ i exponentialfördelningen och beräkna a-posteriori-fördelningen för θ givet observationerna. Ta t ex likformig fördelning $R(0, 30)$ och någon annan fördelning som a-priori-fördelning.

6 Övningsuppgifter om Simulated annealing

Uppgiften består i att med hjälp av simulated annealing lösa det överbestämda binära ekvationssystemet $Ax = b$ där b är en binär $n \times 1$-vektor (alltså bestående av 0:or och 1:or) och A är en binär $n \times m$-matris. Lösningen x är också en binär $m \times 1$-vektor.

Matrismultiplikationen Ax sker modulo 2 dvs resultatet blir i sig en binär vektor.

För att hitta en "lösning" $\widehat{x}$ skall man välja x binär så att Ax överensstämmer med b i så många komponenter som möjligt, dvs $\widehat{x}$ minimerar

$$|Ax - b|, \text{ där } |\cdot| \text{ är summation av absolutbeloppen}$$

Simulated annealing för detta problem utförs genom funktionen *anneal* (alternativt *Eanneal*) som finns att ladda ner från kurshemsidan.

Man behöver generera en b-vektor och en A-matris och det görs t ex genom kommandona

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

där $n > m$, dvs systemet är överbestämt.

Lämpliga värden kan vara $n = 100$ och $m = 50$ samt $delta = 0.1$. Den sistnämnda parametern *delta* är andelen 1:or i A-matrisen och ett för högt värde på denna gör minimeringsproblemet för svårt eftersom $|Ax - b|$ då blir kraftigt diskontinuerlig.

Jag har tillverkat några förberedda A-matriser och b-vektorer som finns lagrade i $an4.mat$ och $an4ny.mat$ i directoriet

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

där de kan laddas ner till Din maskin. Bli inte förbluffad om Netscape försöker öppna dessa binära filer utan spara dem till Din hårddisk. De kan sedan laddas in i Matlab genom kommandot *load filnamn*. Eftersom filerna även finns i $/info/fysikkurser/matstat/funk/$ så finns de tillgängliga genom att man exekverat $efronmain$ efter inloggning i Matlab.

Med kommandot *load an4* får man alltså en b-vektor och en A-matris med $n = 100$ och $m = 50$ och $delta = 0.1$ där jag fått minimivärdet till 16. Jag vet inte om detta är minimum men problemet är för stort (finns $2^{50} \approx 1.13 \cdot 10^{15}$ tänkbara x-vektorer) för att lösa exakt med de metoder jag kan.

Genom att ladda *an4ny* får man en b-vektor av storlek 50×1 och en A-matris av storlek 50×25 där det alltså finns $2^{25} \approx 33.000.000$ tänkbara x-vektorer. (Jag har undersökt detta problem och det finns där en unik lösning som ger värdet 6, dvs med denna lösningsvektor $\widehat{x}$ stämmer alla ekvationer utom 6. Det tog ca 1 dygn att gå igenom alla tänkbara vektorer på en Pentium!)

Använd antingen de "förberedda" problemen (eller generera själv A-matris och b-vektor) samt kör simulated annealing med hjälp av *anneal* - titta på instruktionerna med hjälp av *help anneal*!

7 Användning av programvaran BUGS

För UNIX:

Ordna först så att Du kan köra BUGS genom att

1) antingen låta directoriet $/info/fysikkurser/matstat/bugs/$ ingå i Din path

2) eller kopiera de exekverbara filerna i det directoriet till något directory som ingår i Din path (ett lämpligt ställe är */bin* där man brukar lägga programfiler). Dessa filer heter

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

Dessutom finns en fil *backbugs* som gör att man kan köra *BUGS* som en bakgrundsprocess.

Ladda ner programfilen *pump.bug*, indatafilen *pump.dat* samt initieringsfilen *pump.in* från kurshemsidans *BUGS*-filsdirectory
$http://www.math.kth.se/matstat/gru/5b1555/BUGS/$.
Filerna finns också i directoriet

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

så de kan också kopieras därifrån. (Där finns även filer *dyes.bug* och *dyes*2.*bug* som är exempel på ensidig respektive tvåsidig variansanalys.)

Titta på filerna och försök förstå syntaxen. Kom ihåg att *BUGS* uppdaterar de olika parametrarna med hjälp av Gibbs-sampling, dvs uppdaterar dem successivt med betingade fördelningen för den aktuella parametern givet alla övriga. *BUGS* kräver i princip att dessa är konjugerade, log-konkava eller diskreta - annars kommer *BUGS* att protestera!

pump.bug beskriver samma exempel som presenterades på första föreläsningen, dvs 10 pumpar har körts olika tider och antalet fel på varje har noterats. Modellen är att antalet fel på pump nr i (som körts tiden t_i) är $\text{Po}(\lambda t_i)$. Som aprioriförderlning för λ har man $\Gamma(\alpha, \beta)$ där i sin tur α är $\text{Exp}(1)$ och β är $\Gamma(0.1, 1)$.

Manual för *BUGS* finns på internet på
http : *//www.mrc* − *bsu.cam.ac.uk/bugs/*
där man sedan klickar på *Documentation* och sedan på *Bugs manual v*0.5

En uppsättning exempel finns också tillgänglig. Genom att ladda ned filen *exdir*05.*tar* från

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

och packa upp den genom kommandot

$$tar\ xvpf\ exdir05.tar$$

skapas en hel directory-struktur (i det aktuella directoriet) med exempel. Denna directory-struktur finns även i */info/fysikkurser/matstat/bugs/*.

För DOS laddar man ner *exdir*05.*exe* och packar upp den med kommandot *exdir*05 -o -d. Dessa exempel kan vara värda att titta på!

Gå in i *BUGS* genom kommandot *bugs*. Man kommer ur med *q*().

Kompilera programmet *pump.bug* genom att ge kommandot
compile(“*pump.bug*″).
Låt sen gärna kedjan uppdateras (t ex) 100 steg med kommandot *update*(100) så att “minnet” av de ganska godtyckliga initieringsvärdena “glöms bort”.

Med kommandona *monitor*(*alpha*) och *monitor*(*beta*) meddelar man att man vill få ut variablerna *alpha* och *beta*

Med kommandot *update*(1000) meddelar man att man vill ha 1000 observationer från aposteriorifördelningen för (α, β). När man sedan går ur *BUGS* med *q*() så skapas filen *bugs.out* som innehåller dessa observationer. Notera att man dessutom får en kolumn med löpnummer på observationen. Notera vidare att först kommer α-observationerna och sen β-observationerna.

Ta in dessa utdata i Matlab genom kommandot *load bugs.out*. Tänk på hur filen *bugs.out* är organiserad! Information om detta finns faktiskt i en fil med namnet *bugs.ind* som *BUGS* genererar.

Plotta fördelningarna för α respektive β. Ett bra sätt att illustrera aposteriorifördelningen är att ge den s k prediktiva fördelningen för en ny observation för λ (dvs för en ny pump) genom att generera en $\Gamma(\alpha, \beta)$-observation för varje (α, β)-observation genom att använda Matlab-proceduren *gamrnd*. Notera dock att Matlab låter andraparametern i Γ-fördelningen stå för skalparametern (dvs $1/\beta$).