Exercise E: Metropolis-Hastings

Benedikt Ehinger

6. Januar 2017

Videos K,L,M: MCMC
Recommended Reading Material:

- DBDA Chapter 7
- Statistical Rethinking Chapter 8

1 MCMCs

1.1 Metropolis

We are interested in the posterior $P(\theta|data)$. Our model/likelihood consists of a simple binomial distribution with a single parameter θ. For now we use a flat prior, it is the same value everywhere, i.e. we ignore the prior. The data are:

```
data = c(0,1,1,0,1,1,1,0,1,1)
```

Start with a single step of the Metropolis hasting algorithm: The evaluation of $\frac{f(x^*)}{f(x)}$ with the likelihood-function f, $f(x) = \prod dbinom(data, 1, x)$[1] If the likelihood of $f(x^*)$ is greater, we always take x^* as our new sample, if not, we take x with the probability equal to the ratio. This, simplified, results in:

$$p(take - x^* - instead - of - x) = min(1, \frac{f(x^*)}{f(x)})$$

To evaluate this, sample a uniform random number and check whether it is smaller than the probability above. If yes, accept the new x^* else stay with x for this iteration.

Use a starting value for your MCMC-chain of $\theta = 0.5$. Repeat the whole process for $nIter = 10^6$ iterations.

Plot the whole mcmc-chain and a histogram of the values. In what range do we get the max $P(\theta|data)$? If you want, try out different data-vectors and observe the results.

[1] you can also use $f(x) = dbinom(sum(data), length(data), x)$ which is more efficient but does not make the multiplication explicit.

1.2 Hamiltonian Monte Carlo using Stan - Introduction

We are starting to reach the truly interesting parts of bayesian parameter estimation. We will use a softwarepackage called *'STAN'*. You can get it under `http://mc-stan.org`. The algorithm has one major drawback: Discrete parameters are (currently) only partially supported.

If you use *R* simply run

```
install.packages("rstan")
```

If you use matlab, follow the tutorial online. You need to install the command-line tool "cmdstan"which is then called by acompanying matlab scripts.

For python run:

```
pip install pystan
```

In order to use *STAN* we need to specify our model in the STAN-Language. STAN allows you to use all kinds of programming blocks to define the model (e.g. for, while, if etc.) but in many cases this is not even needed. An example model for a simple gaussian distribution would go as follows:

```
model.stan='
data{                 # block specifies all the variables that are given
  int<lower=0> N;     # number of datapoints
  real y[N];          # list of datapoints (size N)
}
parameters{
  real mu;            # the parameter to be estimated e.g. p(mu|data)
}
model{
  y ~ normal(mu,1);   # y is sampled from a normal
                      # with unknown mu and sigma = 1
}
'
```

First we define a *data* block, all variables from your R/Python/Matlab environment need to go in here. You need to specify the type, if you can the range (e.g. `<lower=0,upper=10>`) and whether it is a single number or a list, as can be seen in thee xample.

The next block is *parameters*. Here you specify all new variables that you want to estimate using MCMC. In the exercise before, this would be θ. But here it would be μ of a normal distribution.

The last block (for now, there are some others) is *model*. Here you specify how parameters relate to each other. In our example there is only a single parameter which parameterizes the mean of a normal distribution. But you can imagine that you want to estimate the σ of the distribution as well.

Let's call the model (either saved in a new file or as a string) with a four separate MCMC-chains and 1000 iterations each.

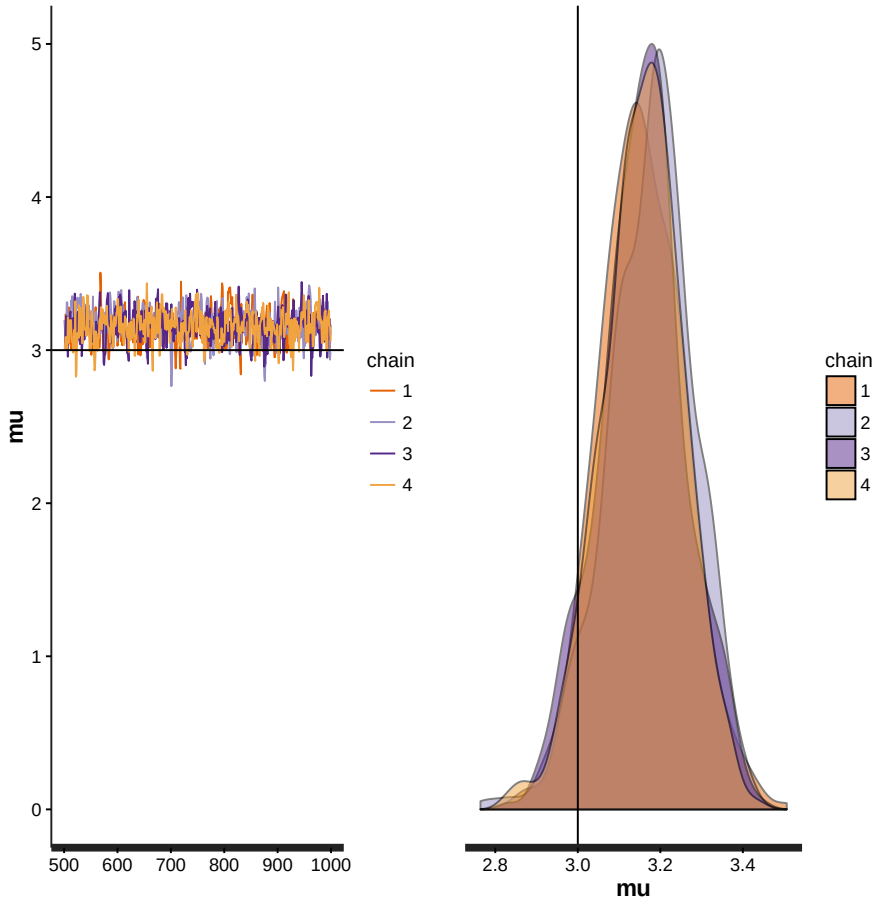
```
library(rstan)

## Loading required package: ggplot2
## rstan (Version 2.9.0-3, packaged: 2016-02-11 15:54:41 UTC, GitRev:
05c3d0058b6a)
## For execution on a local, multicore CPU with excess RAM we recommend
calling
## rstan_options(auto_write = TRUE)
## options(mc.cores = parallel::detectCores())

data = rnorm(100,3,1)
m = stan(model_code = model.stan,data=list(N = length(data),y = data),chains=4,iter=1000)

##
## SAMPLING FOR MODEL 'c3ff9a34b65be6baa4e18193a4843c77' NOW (CHAIN 1).
##
## Chain 1, Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1, Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1, Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1, Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1, Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1, Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1, Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1, Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1, Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1, Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1, Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1, Iteration: 1000 / 1000 [100%]  (Sampling)#
## #  Elapsed Time: 0.004 seconds (Warm-up)
## #                0.004 seconds (Sampling)
## #                0.008 seconds (Total)
## #
##
## SAMPLING FOR MODEL 'c3ff9a34b65be6baa4e18193a4843c77' NOW (CHAIN 2).
##
## Chain 2, Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2, Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2, Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2, Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2, Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2, Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2, Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2, Iteration: 600 / 1000 [ 60%]  (Sampling)
```

```
## Chain 2, Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2, Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2, Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2, Iteration: 1000 / 1000 [100%]  (Sampling)#
## #  Elapsed Time: 0.004 seconds (Warm-up)
## #                0.004 seconds (Sampling)
## #                0.008 seconds (Total)
## #
##
## SAMPLING FOR MODEL 'c3ff9a34b65be6baa4e18193a4843c77' NOW (CHAIN 3).
##
## Chain 3, Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3, Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3, Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3, Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3, Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3, Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3, Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3, Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3, Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3, Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3, Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3, Iteration: 1000 / 1000 [100%]  (Sampling)#
## #  Elapsed Time: 0.004 seconds (Warm-up)
## #                0.004 seconds (Sampling)
## #                0.008 seconds (Total)
## #
##
## SAMPLING FOR MODEL 'c3ff9a34b65be6baa4e18193a4843c77' NOW (CHAIN 4).
##
## Chain 4, Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4, Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4, Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4, Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4, Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4, Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4, Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4, Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4, Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4, Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4, Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4, Iteration: 1000 / 1000 [100%]  (Sampling)#
## #  Elapsed Time: 0.004 seconds (Warm-up)
## #                0.003 seconds (Sampling)
## #                0.007 seconds (Total)
```

```
## #
```

This simple model is quite fast. Let's look at the MCMC chains for our single parameter.

```
cowplot::plot_grid(traceplot(m)+ylim(c(0,5))+geom_hline(yintercept=3),
                   stan_dens(m,separate_chains=T)+geom_vline(xintercept=3))
```



The values for the chains overlap quite nicely, the MCMC chain converged. Notice that only the last half of our iterations have been plotted, the first 500 iterations have been discarded. These are part of the *warmup-period*. In this period STAN tries to estimate parameters to efficiently sample the posterior space.

1.3 HMC Exercise

1. Modify the normal-distribution example so that it fits the binomial data from the previous exercise. What you will need is a different distribution:

here we want to make use of the *bernoulli* distribution. As expected this distribution only has a single parameter that is unknown. Be sure to define a proper range for the parameter and adapt the input data accordingly.

2. Plot the chains and the density similarily to the example. Compare it to exercise 1.

3. Have a look at the RHat statistic, what does it do and why is it useful?

4. We want to add a prior with a truncated normal with $\mu = 0.5$ and $\sigma = 0.1$ (you could plot it using "dnorm"[R] or "normpdf"[matlab]. Add the following line to a new model definition in the "modelblock:

```
  'theta ~ normal(0.5,0.1); # Defines that mu is sampled from a normal
                            # In STAN this specifices the prior for mu
                            # it is assumed truncated in stan because you
                            # defined a lower & upper bound in the
                            # "parameter"-block, haven\'t you?
  '
}
```

   Rerun the estimation and plot the posterior. Has the posterior estimate changed adequately?

5. Have a look at the autocorrelation function of your chains ('stan_ac' from the rstan package or use the 'acf' function in R). What does it tell you? How does STAN compare to your metropolis-random walk from the previous exercise?