Mattias Villani
Avdelning för statistik
Institutionen för datavetenskap
Linköpings universitet
mattias.villani@liu.se

2012-12-04

Bayesian Learning, 6 hp

Computer lab 4

You can use any programming language for the labs, but my hints, help and solutions will be in R.
You are allowed to work and submit your labs in pairs, but do make sure that both of you are contributing.
The **deadline** for this lab is **December 17**.

1. *The Metropolis algorithm*

   (a) Program a general function that uses the Metropolis algorithm to generate random draws from an arbitrary probability density. We will here assume that the probability density is a posterior density from some parameter. The user of your function should be able to supply her own posterior density function and still be able to use your Metropolis function. This is not so straightforward, unless you have come across *function objects* in R and the triple dot (...) wildcard argument. Let me give you the roadmap. Ok, take a deep breath ...
   First, one of the input arguments of your Metropolis function should be `logPostFunc` (or some other suitable name). `logPostFunc` is a *function object* that computes the log posterior density at any value of the parameter vector. This is needed when you compute the acceptance probability of the Metropolis algorithm. I suggest always to program the *log* posterior density, since logs are more stable and avoids problems with too small or large numbers (overflow). Note that the ratio of posterior densities in the Metropolis acceptance probability can be written

   $$\frac{p(\theta^*|y)}{p(\theta^{(t-1)}|y)} = \exp\left[\log p(\theta^*|y) - \log p(\theta^{(t-1)}|y)\right]$$

   This is smart since the large or small common factors in $p(\theta^*|y)$ and $p(\theta^{(t-1)}|y)$ cancel out before we evaluate the exponential function (which can otherwise overflow).
   Second, the first argument of your (log) posterior function should be `theta`, the (vector) of parameters for which the posterior density is evaluated. You can of course use some other name for the variable, but it must be the *first* argument of the posterior density function.
   Third, the user's posterior density is also a function of the data and prior hyperparameters and those need to be supplied to the Metropolis function in some way. The nice way of doing that is use the triple dot (...) argument which is like a wildcard for *any* parameters supplied by the user. This makes it possible to use the Metropolis function for any problem, even when you

as a programmer don't know what the user's posterior density function looks like or what kind of data and hyperparameters being used in that particular problem. To make all of this very clear (I hope!) I give a simple code below where the log posterior density for the Bernoulli model with a Beta prior is coded. That log posterior density is then used in a useless, but illustrative, function `MultiplyByTwo` that returns 2 times the log posterior density evalated at $\theta = 0.3$. Note how the `MultiplyByTwo` takes a function object as input and how it uses the triple dot $(\ldots)$ argument to supply the data s and f, and the prior hyperparameters a and b without explicitly using these symbols inside the function. That makes the useless `MultiplyByTwo` function applicable for *any* function.

Matlab users: Here is one instance where Matlab's lack of proper function objects becomes a pain. You can do the same thing with `eval()` and `varargin`, however, but it is not as pretty as in R or Python. See the Matlab help.

(b) Use your Metropolis function from a) to simulate from the posterior θ of the Bernoulli model with the data $s = 10$, $f = 3$. Use a $\theta \sim Beta(3, 3)$ prior.

(c) Use your Metropolis function to simulate from the posterior of the β vector in the logistic regression model in the spam data from Lab 3. Compare with the normal approximation obtained in Lab 3, Exercise 2a. Don't forget than my code OptimizeSpamR.zip computes the observed information matrix and contains the code for the log posterior of the logistic regression (re-use it!).

(d) [Bonus question for PhD students]. Use your Metropolis function to simulate from the posterior of the β vector in the probit regression model in the spam data from Lab 3. Compare the MCMC efficiency of your Metropolis draws to the draws from your Gibbs sampler in Lab 3, Exercise 2c. If you fail to compute the observed information matrix (the numerical optimization is tricky here), you can try with some other covariance matrix in the proposal density, perhaps an identity matrix or something smarter.]

May Bayes be with you!

```r
# This is the log posterior density of the beta(s+a,f+b) density
LogPostBernBeta <- function(theta, s, f, a, b) {
    logPost <- (s + a - 1) * log(theta) + (f + b - 1) * log(1 - theta)
    return(logPost)
}

# Testing if the log posterior function works
s <- 8
f <- 2
a <- 1
b <- 1
logPost <- LogPostBernBeta(theta = 0.1, s, f, a, b)
print(logPost)

## [1] -18.63


# This is a rather useless function that takes the function myFunction,
# evaluates it at x = 0.3, and then returns two times the function value.
MultiplyByTwo <- function(myFunction, ...) {
    x <- 0.3
    y <- myFunction(x, ...)
    return(2 * y)
}
MultiplyByTwo(LogPostBernBeta, s, f, a, b)

## [1] -20.69
```