STA 410/2102, Fall 2014 — Assignment #3

Due at the start of class on December 2. Please hand it in on 8 1/2 by 11 inch paper, stapled in the upper left, with no other packaging.

This assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own. In particular, you should not leave any discussion with someone else with any written notes (either paper or electronic).

In this assignment, you will use Markov Chain Monte Carlo to sample from the posterior distribution for a Bayesian regression model with residuals modeled by a t distribution, using the Metropolis algorithm. Using a t distribution for the residuals prevents extreme points from having a large effect on the regression fit, if the degrees of freedom parameter of the t distribution is small.

The data consists of n pairs of covariate and response values, $(x_1, y_1), \ldots, (x_n, y_n)$. The covariates are considered to be known values, whose distribution is not modeled. Responses are modeled as

$$y_i \;=\; a \;+\; bx_i \;+\; \sigma\epsilon_i$$

with ϵ_i having the t distribution with ν degrees of freedom, independently for each i.

The model parameters we will use are a, b, $s = \log\sigma$, and $d = 1/\sqrt{\nu}$. The a, b, and s parameters have ranges from minus infinity to plus infinity, but d is restricted to the interval $(0, 1)$, which gives a range for $\nu = 1/d^2$ of one to infinity.

Note that the t distribution with ν degrees of freedom has density function

$$f_\nu(x) \;=\; \frac{\Gamma((\nu+1)/2)}{\sqrt{\nu\pi}\,\Gamma(\nu/2)} \left(1 + x^2/\nu\right)^{-(\nu+1)/2}$$

You should write a function `log_likelihood` that takes as arguments a parameter vector (in the order a, b, s, and d) and data vectors for x and y (the same length), and returns the log of the probability of y given x and the parameter values passed. The density for a t distribution is computed by R's built-in `dt` function. You can get the log of this density with `dt(x,nu,log=TRUE)`. Similarly, `dnorm(x,mu,sigma,log=TRUE)` gives the log of a normal density.

You should use a prior distribution in which, a, b, s, and d are independent, with

$$\begin{aligned} a &\sim N(0, 100^2) \\ b &\sim N(0, 100^2) \\ s &\sim N(0, 10^2) \\ d &\sim U(0, 1) \end{aligned}$$

This prior is suitable if there is little knowledge regarding a, b, and $\sigma = e^s$, and we wish for $\nu = 1/d^2$ to have a fairly high probability of taking on either a large value, giving a t distribution that is close to normal, or a small value that makes the tails of the t distribution be much heavier than for a normal distribution.

You should write a function `log_prior` that takes a parameter vector as argument, and returns the log of the prior probability of the parameters (`-Inf` for values out of range).

You should then write a `log_posterior` function that uses `log_prior` and `log_likelihood` to compute the log of the posterior density, plus an arbitrary constant (ie, you do not need to compute the normalizing constant of the posterior distribution).

You should write a `metropolis` function that uses your `log_posterior` function to sample from the posterior distribution of the parameters given the data, using a random-walk Metropolis method. It should take as arguments the data vectors, x and y, an initial value for the parameter vector, a standard deviation for the proposal distribution, and the number of Markov transitions to do. It should return a list with the following elements:

`params`	A matrix of parameter values
`accept`	A logical vector indicating which proposals were accepted
`log_posterior`	The log of the posterior density at each iteration (no normalizing constant)

The `params` matrix will have one row for each MCMC iteration, and one column for each parameter. The proposal distribution for the random-walk Metropolis algorithm should propose to change each parameter independently, drawing a proposed value for each parameter from the normal distribution with mean equal to the value of the parameter in the current state, and standard deviation equal to the proposal standard deviation argument of `metropolis`.

You should apply your Metropolis sampling function to two datasets generated as follows:

```
set.seed(1)
n1 <- 25
x1 <- runif(n1,-1,1)
y1 <- 0.1 + 3.7*x1 + 0.3*rt(n1,4.2)

set.seed(2)
n2 <- 50
x2 <- runif(n2,0,1)
y2 <- -0.9 + 2.1*x2 + 0.1*rt(n2,1.3)
```

For each of these datasets, you should find suitable initial values for the parameters, and a suitable proposal standard deviation, and run your `metropolis` function for as long as needed to get a good sample from the distribution (which may be tens of thousands of iterations). You should also decide how many "burn-in" iterations should be discarded from the beginning of the run, as not being representative of the posterior distribution. The `accept` and `log_posterior` elements of the result returned by `metropolis` may help in doing these things.

For each dataset, you should then produce the following plots:

- Point plots of each parameter at every tenth iteration (including burn-in iterations).
- Pairwise scatterplots of each parameter versus each other parameter, for every tenth iteration excluding burn-in iterations. (You can get these by passing `plot` the result of converting your matrix of parameter values to a data frame using `as.data.frame`.)
- Plots of the regression lines defined by the a and b values in every tenth (or possibly fewer) iteration (excluding burn-in iterations), along with the data points (in red), and the least-squares regression line found with R's `lm` function (in green).

Discuss what these plots say about the statistical effect of using a model with t-distributed residuals, and what they say about how well the Metropolis sampling works for this problem.

You should also predict the response in a new case where the covariate value is $x = 0.9$, by averaging the expected response value found using values of the parameters that you sampled (excluding burn-in iterations), and compare this prediction with that found using `lm`. Make this prediction using several runs of your `metropolis` function with different settings of the random seed, and discuss how variable the results are.