HW6

(a) Assuming a noninformative uniform prior on the parameters in question (α, β), the posterior is $\propto p(y_i|\alpha, \beta, n_i, x_i) \propto [\text{logit}^{-1}(\alpha + \beta x_i)]^{y_i}[1 - \text{logit}^{-1}(\alpha + \beta x_i)]^{n_i - y_i}$. For the purposes of HMC, we'll use the log posterior, which is given by $y_i((\alpha + \beta x_i) - \log(1 + e^{\alpha+\beta x_i})) - (n_i - y_i)\log(1 + e^{\alpha+\beta x_i}) = y_i(\alpha + \beta x_i) - n_i \log(1 + e^{\alpha+\beta x_i})$. In code:

```
inv.logit <- function(x){
  exp(x)/(1+exp(x))
}

log_p_th <- function(th, y, n, x){
  alpha <- th[1]
  beta <- th[2]
  log_prior <- 0
  log_likelihood <- sum(log((inv.logit(alpha + beta * x) ^ y)
                            * (1 - inv.logit(alpha + beta * x)) ^ (n-y)))
  return (log_likelihood + log_prior)
}
```

We can now get the gradients $\frac{d \log p(\alpha,\beta|y_i,n_i,x_i)}{d\alpha} = y_i - n_i \frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}$ and $\frac{d \log p(\alpha,\beta|y_i,n_i,x_i)}{d\beta} = y_i x_i - n_i x_i \frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}} = x_i(y_i - n_i \frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}$. In code:

```
gradient_th <- function(th, y, n, x){
  alpha <- th[1]
  beta <- th[2]
  d_alpha <- sum(y - n * inv.logit(alpha + beta * x))
  d_beta <- sum(x * (y - n * inv.logit(alpha + beta * x)))
  return (c(d_alpha, d_beta))
}
```

Now we want to check the gradient numerically:

```
gradient_th_numerical <- function(th, y, n, x){
  d <- length(th)
  e <- .0001
  diff <- rep(NA, d)
  for (k in 1:d){
    th_hi <- th
    th_lo <- th
    th_hi[k] <- th[k] + e
    th_lo[k] <- th[k] - e
    diff[k] <- (log_p_th(th_hi, y, n, x) - log_p_th(th_lo, y, n, x))/(2 * e)
  }
  return (diff)
}

setwd("~/Documents/BDA/Homework 6")
bioassay <- read.table("bioassay_data.txt", header=TRUE)
x <- bioassay$x
y <- bioassay$y
```

```
n <- bioassay$n
gradient_th(c(1, 1), y, n, x)
```

```
## [1] -4.868  3.582
```

```
gradient_th_numerical(c(1, 1), y, n, x)
```

```
## [1] -4.868  3.582
```

HMC Code:

```
library(rstan)
```

```
## Loading required package: Rcpp
## Loading required package: inline
##
## Attaching package: 'inline'
##
## The following object is masked from 'package:Rcpp':
##
##     registerPlugin
##
## rstan (Version 2.4.0, packaged: 2014-07-21 17:16:17 UTC, GitRev: c995bc8b9d67)
```

```
library(arm)
```

```
## Warning: package 'arm' was built under R version 3.1.3
```

```
## Loading required package: MASS
## Loading required package: Matrix
## Loading required package: lme4
```

```
## Warning: package 'lme4' was built under R version 3.1.1
```

```
##
## arm (Version 1.8-5, built: 2015-05-13)
##
## Working directory is /Users/DavidHalpern/Documents/BDA/Homework 6
##
##
## Attaching package: 'arm'
##
## The following object is masked from 'package:rstan':
##
##     traceplot
```

```
hmc_iteration <- function(th, y, n, x, epsilon, L, M) {
  M_inv <- 1/M
  d <- length(th)
  phi <- rnorm (d, 0, sqrt(M))
  th_old <- th
  log_p_old <- log_p_th (th, y, n, x) - 0.5 * sum(M_inv * phi ^ 2)
  phi <- phi + 0.5 * epsilon * gradient_th(th, y, n, x)
  for (l in 1:L) {
    th <- th + epsilon * M_inv * phi
    phi <- phi + (if (l == L) 0.5 else 1) * epsilon * gradient_th(th, y, n, x)
  }
  phi <- -phi
  log_p_star <- log_p_th(th, y, n, x) - 0.5 * sum(M_inv * phi ^ 2)
  r <- exp(log_p_star - log_p_old)
  if(is.nan(r)) r <- 0
  p_jump <- min(r, 1)
  th_new <- if (runif(1) < p_jump) th else th_old
  return (list (th = th_new, p_jump = p_jump))
}

hmc_run <- function(starting_values, iter, epsilon_0, L_0, M) {
  chains <- nrow(starting_values)
  d <- ncol (starting_values)
  sims <- array(NA, c(iter, chains, d), dimnames = list(NULL, NULL, colnames(starting_values)))
  warmup <- 0.5 * iter
  p_jump <- array(NA, c(iter, chains))
  for (j in 1:chains) {
    th <- starting_values[j,]
    for (t in 1:iter) {
      epsilon <- runif(1, 0, 2 * epsilon_0)
      L <- ceiling(2 * L_0 * runif(1))
      temp <- hmc_iteration(th, y, n, x, epsilon, L, M)
      p_jump[t, j] <- temp$p_jump
      sims[t, j, ] <- temp$th
      th <- temp$th
    }
  }
  monitor (sims, warmup)
  cat("Avg acceptance probs:", fround(colMeans(p_jump[(warmup + 1):iter,]), 2), "\n")
  #return (list(sims = sims, p_jump = p_jump))
}
```

We know from fitting the model in Homework 2 that the standard deviation of alpha is around 1.5 and the standard deviation around beta is around 5.5. So we can try to make M roughly scale with the inverse of the covariance matrix by setting

```
mass_vector <- c(1 / (1.5 ^ 2), 1 / (5.5 ^ 2))
```

Then we set the other parameter values to the recommended values and run the sampler. After playing around with the parameters it seems like you need 250 iterations to get 100 effective samples and $\epsilon = .5$ and $L = 2$ to get the optimal acceptance probabilities:

```r
parameter_names <- c("alpha", "beta")
d <- length(parameter_names)
chains <- 4
starts <- array(NA, c(chains, d), dimnames = list(NULL, parameter_names))
for (j in 1:chains) {
  starts[j,] <- rnorm(d, 0, 15)
}

hmc_run(starting_values = starts, iter = 250, epsilon_0 = .5, L_0 = 2, M = mass_vector)
```

```
## Inference for the input samples (4 chains: each with iter=250; warmup=125):
##
##       mean se_mean  sd 2.5% 25%  50%  75% 97.5% n_eff Rhat
## alpha  1.4     0.1 1.0 -0.4 0.6  1.3  1.9   3.5   116  1.0
## beta  12.4     0.8 6.1  3.9 8.1 11.5 15.5  26.1    65  1.1
##
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
## Avg acceptance probs: 0.66 0.80 0.75 0.80
```