HOMEWORK 6 ANSWERS

5.1a) biasprob
[1] 0.7263436

b) Using m=100000 I get:

acceptrate
[1] 3534
probthetagt0
[1] 0.7328806

c) probthetasirgt0
[1] 0.73406

R Code:

```
#
# Chapter 5 -- Bayesian Computation With R
#                Problem 5.1
#
#  Remove all objects just to be safe
#
rm(list=ls(all=TRUE))
#
library(LearnBayes)
#
#  Set up log function for problem -- In this case the log of
#    the product of product of success probabilites expressed
#    in logit-type form and a Normal Prior with mean 0 and sigma=0.25
#
#  g(theta|y) ~ [exp(y*theta)/(1 + exp(theta))**n]*exp[-(theta - mu)/2*Sigma^2]
#
#      where theta = log[p/(1-p)] => p = exp(theta)/(1 + exp(theta))
#
logf <- function(theta,parameters)
{
   y <- parameters[1]
   n <- parameters[2]
   mu <- parameters[3]
   sigma <- parameters[4]
#
#  log of posterior
#
   logposterior <- y*theta-n*log(1+exp(theta))-((theta-mu)^2)/(2*sigma^2)
   return(logposterior)
}
#
#  laplace is part of the LearnBayes Library -- It finds the mode of the
#     log poserior density.  At the mode it uses a Taylor Series approximation
#     and the posterior density is approximated by a multivariate normal
#     density with mean Theta and VCOV equal to the Inverse numerical Hession
#
#  Note that the second argment is the best guess about the value of theta --
#    theta is the **only** variable here!  Since the data indicate that theta >0
#    we start laplace there to find the mode
#
parameters <- c(5, 5, 0, 0.25)
#
fit <- laplace(logf,0,parameters)
#
#  fit
# $mode
# [1] 0.1449219
```

```
#
# $var -- Note this is the variance, not the standard deviation
#         [,1]
# [1,] 0.057993
#
# $int
# [1] -3.789343
#
# $converge
# [1] TRUE
#
#  So this gives a Normal(0.1449219, 0.057993)
# &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
#  Part (a):  Using the pnorm(x,mean,sd) function in R we get:
#
biasprob <- 1 - pnorm(0,mean=fit$mode,sd=sqrt(fit$var))
#  What this computes is the probability **above** zero for a N(0.1449219, 0.057993)
#
# biasprob
# [1] 0.7263436
#
# &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
#  Part (b): Rejection Sampling
#
#  Need to sample theta from a function p(theta) such that the ratio of
#     the posterior and the sampling function is less than 1:
#
#                 g(theta|y)/[c*p(theta)] < 1 where c is some normalizing constant
#
#  Then draw a uniform random number using runif(x) in R.  If
#
#                 runif(x) <= g(theta|y)/[c*p(theta)] then accept theta
#
#   so (1) draw theta from p(theta)
#      (2) compute value of g(theta|y)/[c*p(theta)]
#      (3) draw uniform random number and accept theta if runif(x) <=
g(theta|y)/[c*p(theta)]
#
#   Note the logic -- the closer the ratio is to one the likelier the acceptance rate.
What
#        this does with a huge number of draws is that it results in a set of thetas that
#        will approximate the posterior distribution.  This will always work provided the
#        p(theta) distribution is ***always above** g(theta|y)
#
#   Simple solution here is to set p(theta) = N(0, sd=.25), namely the prior in the
problem.
#        This **guarantees** that the ratio is less than one
#
rejectsample <- function(m)
{
  theta <- rnorm(m,mean=0,sd=.25)
  ratiogoverp <- exp(5*theta)/(1+exp(theta))^5   # This is just the posterior/prior from
p. 111
  return(theta[runif(m) < ratiogoverp])
}
m <- 100000
partb <- rejectsample(m)
acceptrate <- length(partb)
probthetagt0 <- mean(partb > 0)
#  acceptrate
# [1] 350                              # This is a really low acceptance rate but it
works
#  probthetagt0
# [1] 0.7657143
#
#  Here is a run with m=100,000
#  acceptrate
# [1] 3633
#  probthetagt0
# [1] 0.7263969
#
```

```
# &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
#  part (c): Sampling Importance Resampling (SIR) algorithm
#
#     Similar to (b) but there are four steps:
#   (1) Sample from the proposal density, in our case N(0, sd=0.25), j=1,...m times
#   (2) Compute weights -- posterior/proposal -- w(theta_j) = g(theta_j|y)/p(theta_j)
#   (3) Convert the weights to probabilities -- p_j = w(theta_j)/SUM_j=1,m[w(theta_j)]  -
-
#           Note that this produces m probabilities
#   (4) Use R sample command to draw a sample of the theta_j's with replacement -- The
logic here
#       is that the likelihood of a theta_j being drawn is its ***relative** weight in
the
#       vector of "prob" that is passed to sample.  The larger its weight **relative** to
#       the other weights the more often it is drawn.
#
# sample(x, size, replace = FALSE, prob = NULL) -- the defaults
#
# Arguments for sample function
# x        Either a (numeric, complex, character or logical) vector of more than one
element
#          from which to choose, or a positive integer.
# size     positive integer giving the number of items to choose.
# replace  Should sampling be with replacement? TRUE or FALSE
# prob     A vector of probability weights for obtaining the elements of the vector being
sampled.
#          They need not sum to one, but they should be nonnegative and not all zero.
#
m <- 10000
thetasir <- rnorm(m, mean=0, sd=0.25)
ratiopostprob <- exp(5*thetasir)/(1+exp(thetasir))^5  # ratio of posterior/proposal
probweights <- ratiopostprob/sum(ratiopostprob)
theta.post <- sample(thetasir, size=100000, replace=TRUE, prob=probweights)
probthetasirgt0 <- mean(theta.post > 0)
#
#  probthetasirgt0
#  [1] 0.7298
#
#  Here is a run with m=100,000
#  probthetasirgt0
#  [1] 0.72714
#
```

5.2a)
> theta.interval
[1] 0.07989705 0.93260295
> eta.interval
[1] 0.5199636 0.7176031

b) There are a variety of ways you could have programmed this. Here is what I did:

```
# POL 272 Bayesian Methods
# Assignment 5.2
# Chapter 5, Exercise 2 of Bayesian Computation with R
#
rm(list=ls(all=TRUE))
library(LearnBayes)
#
#  Part (a)
#
mylogpost<-function(eta,data){
       theta <- exp(eta)/(1+exp(eta))
       logpost <- data[1]*log(2+theta)+data[2]*log(1-theta)+data[3]*log(theta)
       return(logpost)
        }
#
data <- NULL
data[1] <- 125
data[2] <- 39
data[3] <- 35
#out<-laplace(mylogpost,mode=1,par=c(125,39,35))
out<-laplace(mylogpost,mode=1,data)
#
# out
# $mode
# [1] 0.50625
# $var
#         [,1]
# [1,] 0.047318
# $int
# [1] 65.32634
# $converge
# [1] TRUE
#
mu<-out$mode
sd<-sqrt(out$var)
#
theta.interval <- mu + c(-1.96, 1.96)*sd
#
# theta.interval
# [1] 0.07989705 0.93260295
#
eta.interval <- exp(theta.interval)/(1+exp(theta.interval))
#
# eta.interval
# [1] 0.5199636 0.7176031
#
#
#  Part (b)
#
#  We are supposed to use a t-distribution with mean and varance from
#    the laplace output above with a small number of degrees of freedom
#    He covers this on pages 99 - 100
#
tparameters <- list(mu = 0.50625, var = 0.047318, df = 4)
#
#  function to compute log(posterior) - log(proposal) -- we use this
#    to find the scaling constant "c" -- see Problem_Chap_5_1.r -- this
#    means that our t-distribution will always be **above** the posterior
```

```
#    used in Part (a) above
#
mylogpostdiff <- function(eta,tparameters){
       theta <- exp(eta)/(1+exp(eta))
       logpostx <- 125*log(2+theta)+39*log(1-theta)+35*log(theta)
#       diff <- logpostx -
dmt(eta,mean=c(tparameters$mu),S=tparameters$var,df=tparameters$df,log=TRUE)
       diff <- mylogpost(eta,data) -
dmt(eta,mean=c(tparameters$mu),S=tparameters$var,df=tparameters$df,log=TRUE)
       return(diff)
}
# Now use laplace to maximize log(posterior) - log(proposal)
#
fmax <- laplace(mylogpostdiff,.5,tparameters)
dmax <- mylogpostdiff(fmax$mode,tparameters)
thetatest <- rejectsampling(mylogpost, tparameters, dmax, 10000, data)
```