HW6\_Reloaded\_R\_code.R

AM

2020-11-12

#source("http://math.ucdenver.edu/~jfrench/data/bayesdists.R")  
#devtools::install\_github("jfrench/bayesutils")  
library(mvtnorm)  
library(bayesplot)

## This is bayesplot version 1.7.2

## - Online documentation and vignettes at mc-stan.org/bayesplot

## - bayesplot theme set to bayesplot::theme\_default()

## \* Does \_not\_ affect other ggplot2 plots

## \* See ?bayesplot\_theme\_set for details on theme setting

library(invgamma)  
library(coda)  
library(rstan)

## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling  
## options(mc.cores = parallel::detectCores()).  
## To avoid recompilation of unchanged Stan programs, we recommend calling  
## rstan\_options(auto\_write = TRUE)

##   
## Attaching package: 'rstan'

## The following object is masked from 'package:coda':  
##   
## traceplot

set.seed(77)  
y = bayesutils::rinvgamma(100, 2, 3)  
range(y)

## [1] 0.4647718 12.0184742

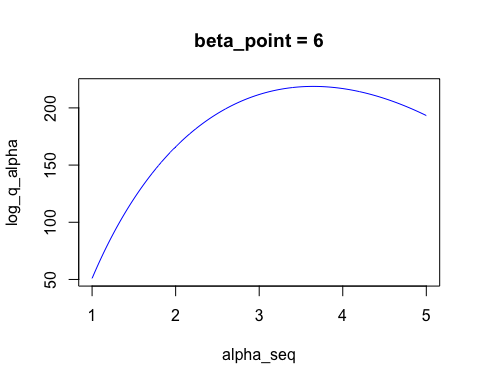
mean(y)

## [1] 2.520916

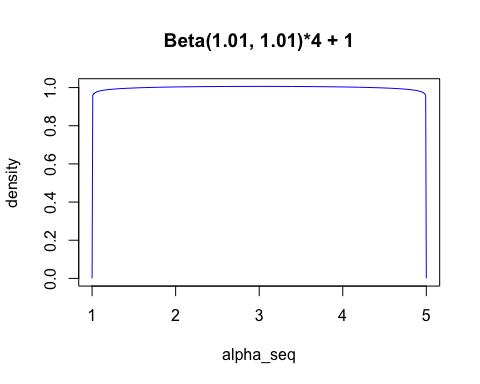
sd(y)

## [1] 2.244544

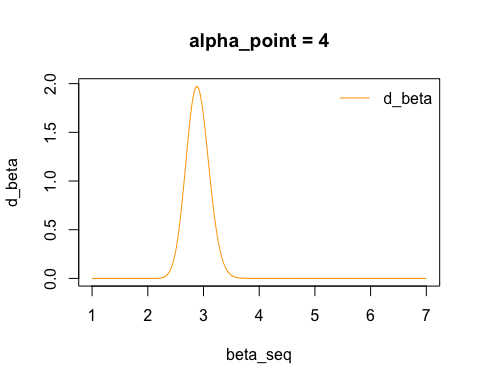
sum\_inv = sum(1/y)  
sigma = 0.3  
beta\_a = 1.01  
beta\_b = 1.01  
### 1b  
log\_q\_alpha = function (a, b) {  
 100\*a\*log(b) +   
 (a+1)\*sum(log(1/y)) -  
 100 \* log(factorial(a-1))  
}  
alpha\_seq = seq(1,5, length=500)  
beta\_point = 6  
plot(alpha\_seq, log\_q\_alpha(a = alpha\_seq, b=beta\_point), ylab = "log\_q\_alpha",   
 type = "l", col = "blue")  
title("beta\_point = 6")



plot(alpha\_seq, dbeta((alpha\_seq-1)/4, 1.01,1.01), ylab = "density",   
 type = "l", col = "blue")  
title("Beta(1.01, 1.01)\*4 + 1")



### 1b  
alpha\_point = 2  
beta\_seq = seq(1, 7, length = 1000)  
  
d\_beta = function(a, b) {  
 dgamma(b, shape = (100\*a + 4), rate = (4 + sum\_inv))  
}  
plot(beta\_seq, d\_beta(a = alpha\_point, b=beta\_seq), ylab = "d\_beta",   
 type = "l", col = "orange")  
legend("topright",   
 legend = c("d\_beta"),   
 col = c("orange"), lty = 1, bty = "n")  
title("alpha\_point = 4")



### 2c  
mh\_gibbs = function(B, theta\_start) {  
 theta = array(0, c((B+1), 2))  
 theta[1,1] = theta\_start[1]  
 theta[1,2] = theta\_start[2]  
  
 for (i in 2:dim(theta)[1]) {  
 alpha\_star = theta[(i-1),1]  
 beta\_star = rgamma(1, shape = (100\*alpha\_star+4), rate = (4 + sum\_inv))  
 theta[i,2] = beta\_star  
 alpha\_star = rbeta(1, beta\_a, beta\_b)\*4+1  
 #alpha\_star = runif(1, 1, 5)  
 #alpha\_star = rnorm(1, theta[(i-1),1], sigma)   
 #while ((alpha\_star < 1)|(alpha\_star > 5)) {  
 # alpha\_star = rnorm(1, theta[(i-1),1], sigma)   
 #}  
 ### why does rnorm work better than runif??  
  
 num\_logr = log\_q\_alpha(a=alpha\_star, b=beta\_star) -  
 dbeta((alpha\_star-1)/4,beta\_a,beta\_b)  
 #dnorm(alpha\_star, theta[i-1, 1], sigma, log = TRUE)   
 #-   
 #dunif(alpha\_star, 1, 5, log = TRUE)   
 ### think it is possible not to correct for the proposal/ jumping (both for unif and norm) ###  
 ### as it is symmetric and thus cancels in den\_logr ###  
 den\_logr = log\_q\_alpha(a=theta[i-1, 1], b=beta\_star) -  
 dbeta((theta[i-1, 1]-1)/4,beta\_a,beta\_b)  
   
 #dnorm(theta[i-1, 1], alpha\_star, sigma, log = TRUE)   
 #-  
 #dunif(alpha\_star, 1, 5, log = TRUE)  
 ### think it is possible not to correct for the proposal/ jumping (both for unif and norm) ###  
 ### as it is symmetric and thus cancels in den\_logr ###  
 logr = num\_logr - den\_logr  
 if (log(runif(1)) <= min(logr, 0)) {  
 theta[i,1] = alpha\_star  
 } else {  
 theta[i,1] = theta[(i - 1), 1]  
 }  
 }  
 return(theta)  
}  
B = 10^5  
keep = (B/2 + 1):(B + 1)  
chain1 = mh\_gibbs(B, theta\_start = c(2.3,4))  
chain2 = mh\_gibbs(B, theta\_start = c(2.8,5))  
chain3 = mh\_gibbs(B, theta\_start = c(3.3,6))  
chain4 = mh\_gibbs(B, theta\_start = c(3.9,8))  
chain5 = mh\_gibbs(B, theta\_start = c(4.2,10))   
  
chain = rbind(chain1[keep,], chain2[keep,], chain3[keep,], chain4[keep,], chain5[keep,])  
mc\_a = mcmc.list(mcmc(chain1[keep,1]), mcmc(chain2[keep,1]),  
 mcmc(chain3[keep,1]), mcmc(chain4[keep,1]),  
 mcmc(chain5[keep,1]))  
summary (mc\_a)

##   
## Iterations = 1:50001  
## Thinning interval = 1   
## Number of chains = 5   
## Sample size per chain = 50001   
##   
## 1. Empirical mean and standard deviation for each variable,  
## plus standard error of the mean:  
##   
## Mean SD Naive SE Time-series SE   
## 2.0114898 0.2498864 0.0004998 0.0039133   
##   
## 2. Quantiles for each variable:  
##   
## 2.5% 25% 50% 75% 97.5%   
## 1.550 1.837 2.003 2.176 2.528

mc\_b= mcmc.list(mcmc(chain1[keep,2]), mcmc(chain2[keep,2]),  
 mcmc(chain3[keep,2]), mcmc(chain4[keep,2]),  
 mcmc(chain5[keep,2]))  
summary (mc\_b)

##   
## Iterations = 1:50001  
## Thinning interval = 1   
## Number of chains = 5   
## Sample size per chain = 50001   
##   
## 1. Empirical mean and standard deviation for each variable,  
## plus standard error of the mean:  
##   
## Mean SD Naive SE Time-series SE   
## 2.9121331 0.4080953 0.0008162 0.0056665   
##   
## 2. Quantiles for each variable:  
##   
## 2.5% 25% 50% 75% 97.5%   
## 2.161 2.626 2.896 3.180 3.757