STA 602 HW5

William Tirone

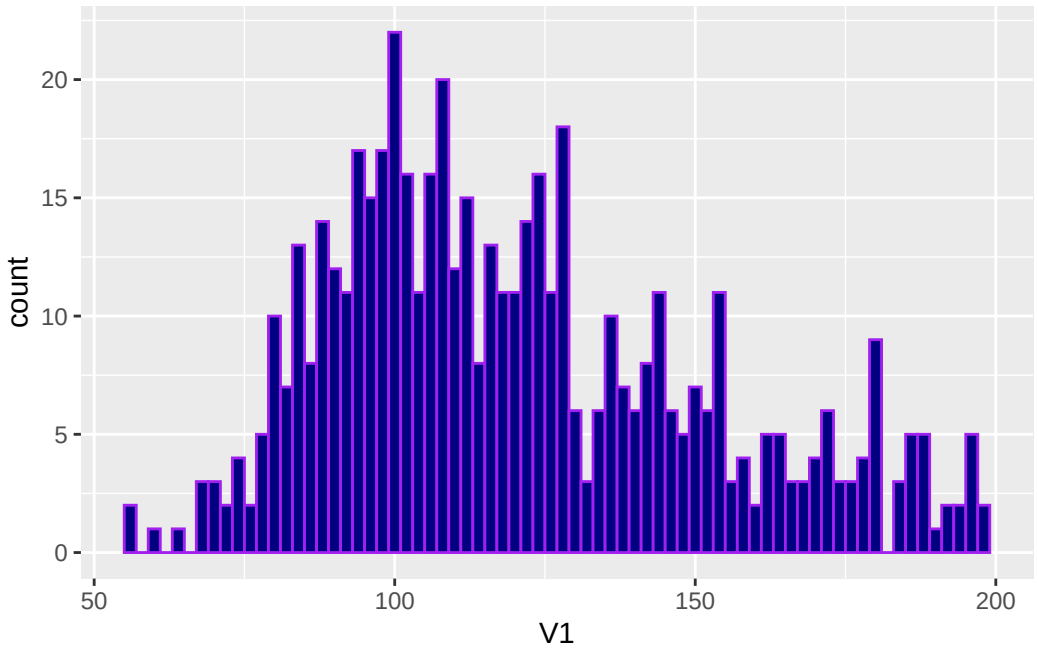
```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.3.6      v purrr   0.3.5
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.2.1      v stringr 1.4.1
v readr   2.1.3      v forcats 0.5.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
```

6.2

a)

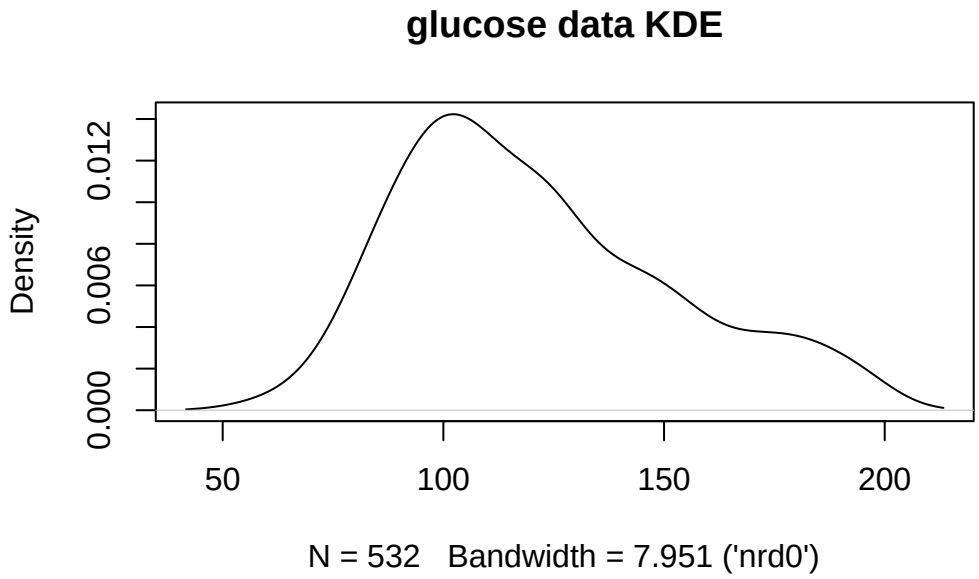
The data is not symmetric, and has more mass in the right tail than a normal distribution would have.

```
glucose = read.table('http://www2.stat.duke.edu/~pdh10/FCBS/Exercises/glucose.dat')
ggplot() + geom_histogram(data=glucose,
                          aes(x=V1),
                          binwidth = 2,
                          col='purple',
                          fill='navy')
```



I don't know anything about KDE, but it looks like this agrees with the histogram approach, more mass in the right tail.

```
kde = kdensity(glucose$V1)
plot(kde, main = "glucose data KDE")
```



part b hand written

c)

Gibb's sampler below does not work. I've worked on it for 10+ hours and can't get it to work, there's some kind of small bug I believe that's producing null values for the x draws. Just want to show some proof that I actually attempted this.

```
# data and starting vals
n = length(glucose$V1)


# priors
a = 1
b = 1
mu0 = 120
tau0.sq = 200
sig0.sq = 1000
nu0 = 10

# iterations
S = 10000

# matrix to store values
# missing p or the data here???
vals = matrix(NA,nrow=S,ncol=4,dimnames=list(1:S,c("theta1","sig1^2","theta2","sig2^2")))
part_c = matrix(NA,nrow=S,ncol=2, dimnames=list(1:S,c("x","y_tilde")))

# part b values
vals.init = c(100,1000,100,1000)
vals.curr = vals.init

# part c values
partc.vals.init = c(1,1) # x , ytilde
partc.vals.curr = partc.vals.init


p = 0.5


for (i in 1:S) {
```

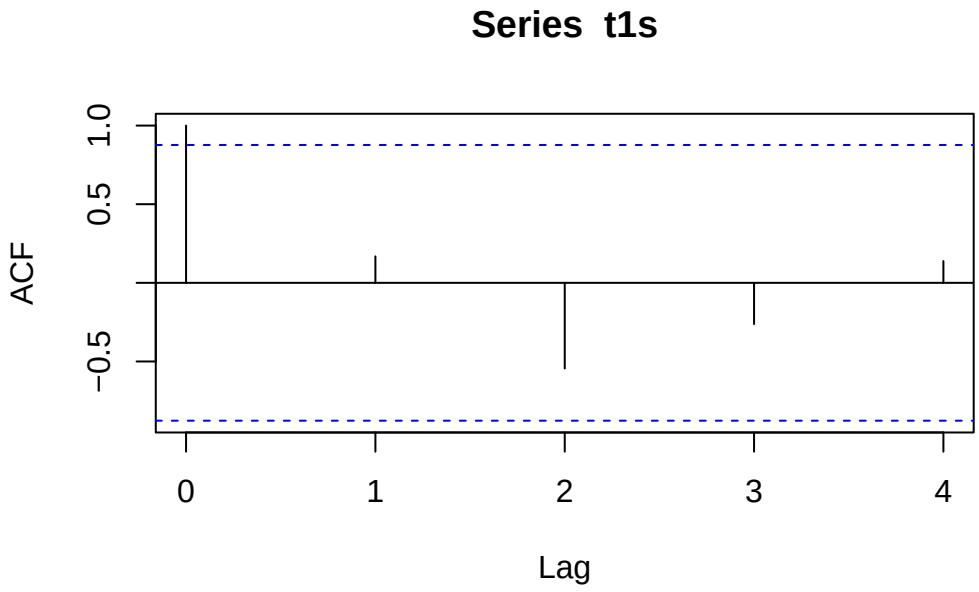
```
t1 = vals.curr[1]
sig.sq.1 = vals.curr[2]
t2 = vals.curr[3]
sig.sq.2 = vals.curr[4]

x = partc.vals.curr[1]
y_tilde = partc.vals.curr[2]

# sampling for x
# discussed with TA, not sure I understand this part
full_condtl_x = (p * dnorm(glucose$V1, t1, sqrt(sig.sq.1))) /
                  (p * dnorm(glucose$V1, t1, sqrt(sig.sq.1)) +
                  (1-p) * dnorm(glucose$V1, t2, sqrt(sig.sq.2)))

# getting null values so filling with p, not correct though
full_condtl_x[is.na(full_condtl_x)] = p

x = rbinom(n,1,full_condtl_x)

# now can split groups with X = 1 or X = 2
n1 = sum(x)
n2 = n - n1

# splitting into two groups
# using 1 and 0 but book used 1 / 2
y.1 = glucose$V1[x == 1]
y.2 = glucose$V1[x == 0]
y.bar1 = mean(y.1)
y.bar2 = mean(y.2)
y.var1 = var(y.1)
y.var2 = var(y.2)

p = rbeta(1, n1 + a, n2 + b)

# update t1
vals.curr[1] = rnorm(1,
                     mean = ((mu0/tau0.sq) + (n1*y.bar1)/vals.curr[2]) /
                       ((1/tau0.sq) + n1/vals.curr[2]),
                     sd = sqrt(1 / ((1/tau0.sq)+(n1/vals.curr[2]))))

# update sig.sq.1
```

```
  vals.curr[2] = 1/ rgamma(1,
                           shape = (nu0 + n1)/2,
                           rate = .5 * (nu0 * sig0.sq + (n1 - 1)*
                                          y.var1 + n1*(y.bar1 - vals.curr[1])^2))

  # update t2
  vals.curr[3] = rnorm(1,
                       mean = ((mu0/tau0.sq) + (n1*y.bar2)/vals.curr[4]) /
                         ((1/tau0.sq) + n2/vals.curr[4]),
                       sd = sqrt(1 / ((1/tau0.sq)+(n2/vals.curr[4]))))

  # update sig.sq.2
  vals.curr[4] = 1/ rgamma(1,
                           shape = (nu0 + n2)/2,
                           rate = .5 * (nu0 * sig0.sq + (n2 - 1)*
                                          y.var2 + n2*(y.bar2 - vals.curr[3])^2))

  # part c, doesn't work

  # x value
  # partc.vals.curr[1] = rbernoulli(p)
  # Y_s = rnorm(1, 0, 1)

  # Y tilde value
  # partc.vals.curr[2] = Y_s

  # saving current iteration
  vals[i,] = vals.curr
  # part_c[i,] = partc.vals.curr

}
```

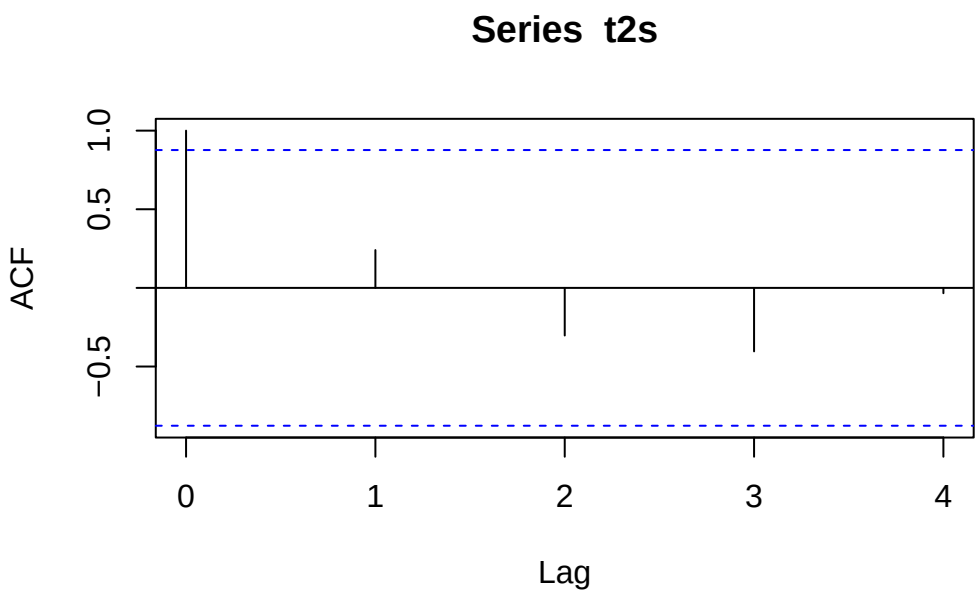
Autocorrelation plots

Code does not work so I just picked the first few rows that actually produced data, the rest produced null values

```
t1s = apply(vals[0:5,c(1,3)], 1, FUN=min)
t2s = apply(vals[0:5,c(1,3)], 1, FUN=max)

acf(t1s)
```




```
acf(t2s)
```




```
effectiveSize(t1s)
```

```
var1
   5
```

```
effectiveSize(t2s)
```

```
var1
   5
```

d)

Included (or attempted...) in code for part c.

6.3

part b and c on hand written attachments.

c)

Borrowed code from this resource and theory from these extra lecture notes. I tried to adapt code that used matrix notation, but honestly don't understand this problem. I don't think we were given enough resources or direction to do this problem effectively.

I left my code in because I wanted to show that I at least tried, but I definitely don't understand it. Though, if I did, I'm pretty sure using the matrix approach below would greatly simplify the Gibb's sampler code, so interested to learn more about that.

```
# data
n = length(divorce$V2)
n1 = sum(divorce$V2) # divorce
n2 = n - n1 # no divorce
X = as.matrix(divorce)

# priors
beta_0 <- rep(0, 2)
Q_0 <- diag(10, 2)

# iterations
S = 10000

# Initialize parameters
beta <- rep(0, 2)
z <- rep(0, n)
```

```r
# Compute posterior variance of beta
prec_0 <- solve(Q_0)
V <- solve(prec_0 + crossprod(X, X))

# samples
beta_chain <- matrix(0, nrow = S, ncol = 2)

# random y's because I don't know what I'm doing
y <- rbinom(n, 1, .5)


for (t in 2:S) {

  # Update Mean of z
  mu_z <- X %*% beta

  # Draw latent variable z from its full conditional: z | beta, y, X
  z[y == 0] <- rtruncnorm(n1, mean = mu_z[y == 0], sd = 1, a = -Inf, b = 0)
  z[y == 1] <- rtruncnorm(n2, mean = mu_z[y == 1], sd = 1, a = 0, b = Inf)

  # Compute posterior mean of beta
  M <- V %*% (prec_0 %*% beta_0 + crossprod(X, z))

  # Draw variable \beta from its full conditional: \beta | z, X
  beta <- c(rmvnorm(1, M, V))

  # Store the \beta draws
  beta_chain[t, ] <- beta
}
```

d)

couldn't complete.