MAST30027: Modern Applied Statistics

Week 12 Lab

1. **Metropolis-Hastings** [**We already solved problems (a), (b), and (c) in the week 10. This week, solve the problem (d), (e), and (f).**]

   Recall that $\mathbf{X} = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with $\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$ and $\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix}$, iff $\mathbf{X}$ has joint density

   $$f_{\boldsymbol{\mu},\boldsymbol{\Sigma}}(\mathbf{x}) = \frac{1}{2\pi|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right),$$

   where $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$.

   (a) Write an R function that evaluates the density of a bivariate normal distribution. The function should take as input the point $\mathbf{x}$, the mean $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$.
   You will find the functions `solve` and `det` useful.
   Solution:

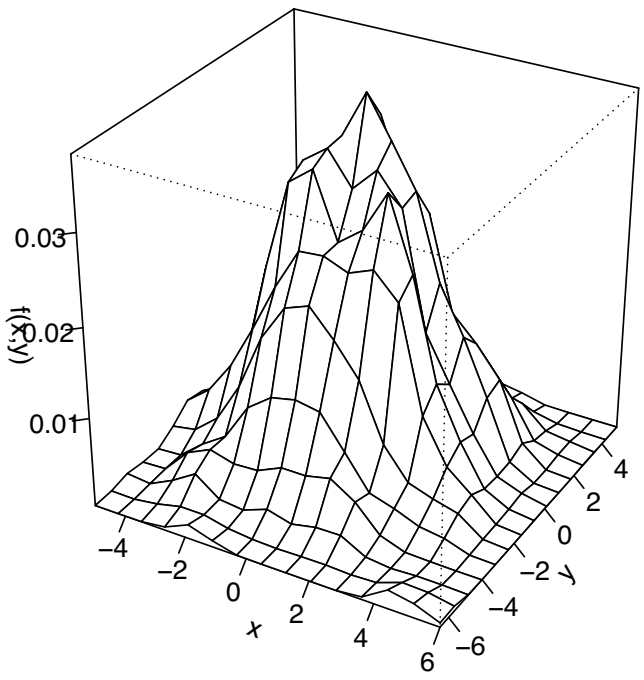
```
> dbinorm <- function(x, mu, Si) {
+   # x and mu are vectors length 2 and Si a 2x2 matric
+   # returns the density at x of a bivariate normal mean mu var Si
+   exp(-t(x - mu)%*%solve(Si, x - mu)/2)/2/pi/sqrt(det(Si))
+ }
```

   You can check that it is working by noting that `dbinorm(c(1,1), c(0,0), matrix(c(1,0,0,1),2,2))` gives the same answer as `dnorm(1)^2`.

   (b) Write a program in R that uses the Metropolis-Hastings algorithm to generate a sample of size $n = 1000$ from the $N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}\right)$ distribution. Use the symmetric random walk proposal distribution $N\left(\mathbf{x}, \sigma^2 I\right)$ with $\sigma = 2.5$.
   Use $\mathbf{X}(0) = \begin{pmatrix} 6 \\ -6 \end{pmatrix}$ as your initial state. Report the proportion of accepted values.
   Solution:

```
> # Metropolis-Hastings simulation of a bivariate normal
> # inputs
> mu <- c(0, 0)                     # mean
> Si <- matrix(c(4, 1, 1, 4), 2, 2) # variance
> iterations <- 1000      # sample size
> startvalue <- c(6, -6)  # initial value
> sd <- 2.5               # std-dev for proposal chain
> # main loop
> chain <- matrix(nrow = iterations+1, ncol = 2)
> chain[1,] <- startvalue
> accepted <- 0 # counts num accepted proposals
> for (i in 1:iterations){
+   proposal <- rnorm(2, chain[i,], sd)
+   prob <- dbinorm(proposal, mu, Si)/dbinorm(chain[i,], mu, Si)
+   if (is.nan(prob)) prob <- 0
+   if (runif(1) < prob) {
+     chain[i+1,] <- proposal
+     accepted <- accepted + 1
+   } else {
+     chain[i+1,] <- chain[i,]
+   }
```

```
+ }
> # acceptance rate
> accepted/iterations
[1] 0.457
> # 2D density plot
> library(MASS)
> chain_density <- kde2d(chain[,1], chain[,2], n = 15)
> persp(chain_density, phi = 30, theta = 30, d = 5,
+       xlab = "x", ylab = "y", zlab = "f(x,y)",
+       ticktype = "detailed")
```



The acceptance rate was 46% (this will vary a little every time you run the program).

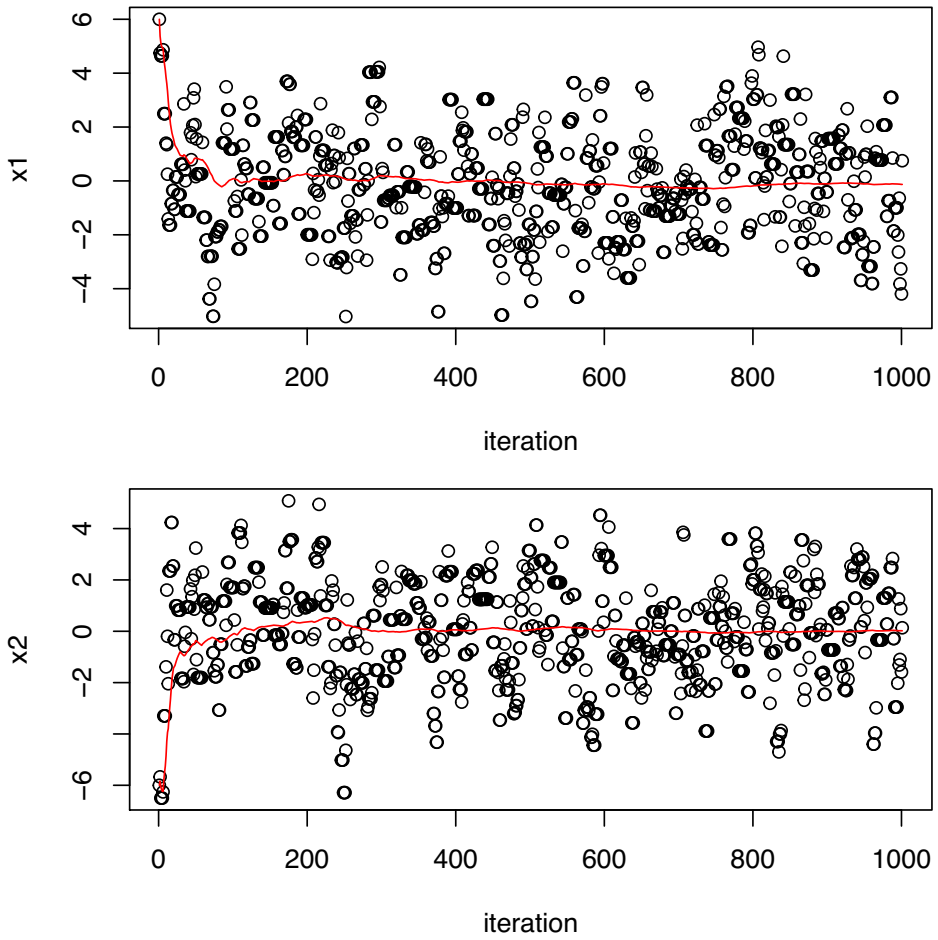
To check that the output looks ok, I have plotted (a kernel density estimate of) the joint density. Easier than plotting a joint density would be to plot histograms/densities of the marginal samples, using `hist` or `density`.

(c) Let $\mathbf{X}(n)$ be the n-th sample point. Plot $X_i(n)$ and the cumulative averages $\overline{X}_i(n) = n^{-1}\sum_{j=1}^{n} X_i(j)$, for $i = 1, 2$. The cumulative averages should give a rough idea of how quickly the $\mathbf{X}(n)$ converge in distribution.

Solution:

```
> # cumulative averages
> par(mfrow=c(2,1), mar=c(4,4,1,1))
> plot(chain[,1], xlab = "iteration", ylab = "x1")
> cumavg1 <- cumsum(chain[,1])/1:(iterations + 1)
> lines(1:(iterations + 1), cumavg1, col = "red")
> plot(chain[,2], xlab = "iteration", ylab = "x2")
> cumavg2 <- cumsum(chain[,2])/1:(iterations + 1)
```

```
> lines(1:(iterations + 1), cumavg2, col = "red")
>
>
```



(d) You can use the R functions `cor` or `acf` to estimate the lag 1 autocorrelation ρ. Calculate the so called *effective sample size*: $n(1-\rho)/(1+\rho)$ for each variable.

Solution:

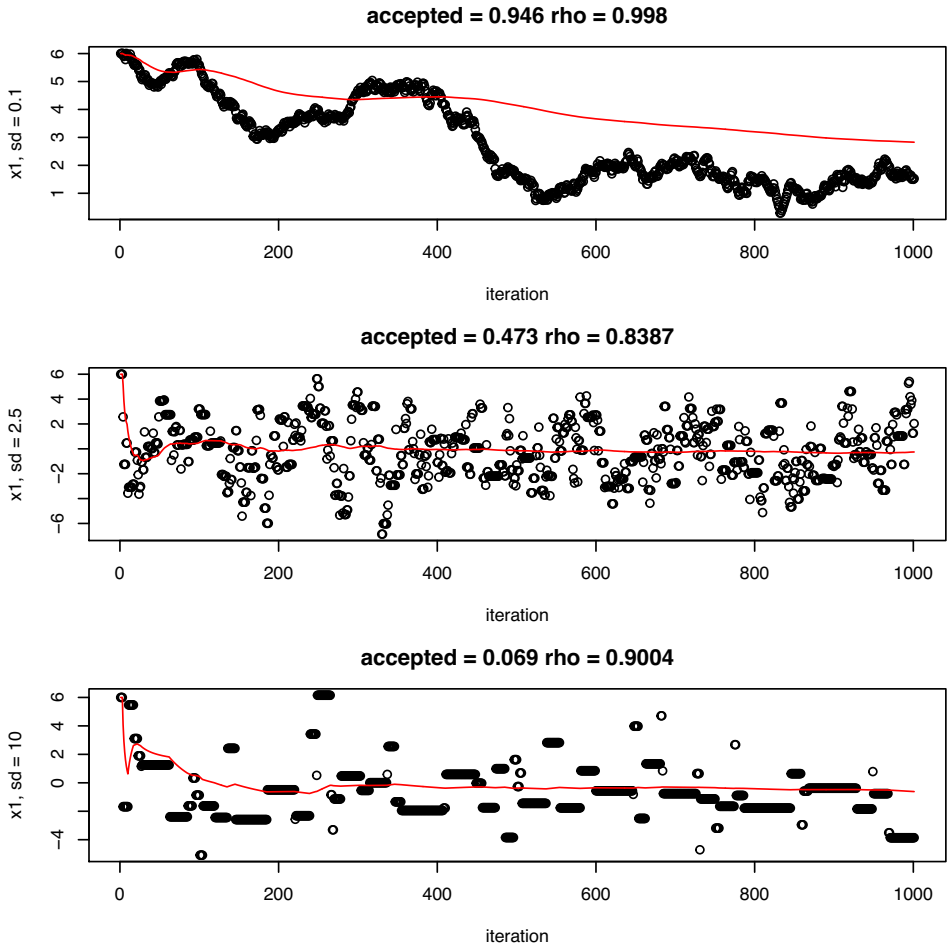
```
> # effective sample size
> (rho1 <- cor(chain[-(iterations+1),1], chain[-1,1]))
[1] 0.7475516
> rho1
[1] 0.7475516
> iterations * (1 - rho1)/(1 + rho1)
[1] 144.4583
> (rho2 <- cor(chain[-(iterations+1),2], chain[-1,2]))
[1] 0.7931476
> rho2
[1] 0.7931476
> iterations * (1 - rho2)/(1 + rho2)
[1] 115.3571
```

For both chains the lag 1 autocorrelation was quite high, at 75% and 79%. This is influenced by both the acceptance rate (lower rate means higher correlation) and the size of steps made by the proposal chain (smaller steps means higher correlation, but smaller steps also generally mean a higher acceptance rate). The effective sample sizes were 144 and 115 (out of 1000).

(e) Change the standard deviation of the proposal chain to $\sigma = 0.1$ and then $\sigma = 10$. How do the proportion of accepted values, the cumulative averages, and the effective sample size change?

Solution: We just give the results for the first co-ordinate. The second co-ordinate behaves similarly.

```
> dbinorm <- function(x, mu, Si) {
+   # x and mu are vectors length 2 and Si a 2x2 matric
+   # returns the density at x of a bivariate normal mean mu var Si
+   exp(-t(x - mu)%*%solve(Si, x - mu)/2)/2/pi/sqrt(det(Si))
+ }
> rbinormMH <- function(mu, Si, iterations = 1000, startvalue = c(6, -6), sd = 2.5) {
+   # mu = mean, Si = variance, startvalue = initial value, sd = std-dev
+   # for proposal chain
+   chain <- matrix(nrow = iterations+1, ncol = 2)
+   chain[1,] <- startvalue
+   accepted <- 0
+   for (i in 1:iterations){
+     proposal = rnorm(2, chain[i,], sd)
+     prob = dbinorm(proposal, mu, Si)/dbinorm(chain[i,], mu, Si)
+     if (is.nan(prob)) prob <- 0
+     if (runif(1) < prob) {
+       chain[i+1,] = proposal
+       accepted <- accepted + 1
+     } else {
+       chain[i+1,] = chain[i,]
+     }
+   }
+   attributes(chain) <- list(dim = c(iterations+1,2),
+                             p_accept = accepted/iterations,
+                             rho1 = cor(chain[-(iterations+1),1], chain[-1,1]),
+                             rho2 = cor(chain[-(iterations+1),2], chain[-1,2]))
+   return(chain)
+ }
> par(mfrow=c(3,1), mar=c(4,4,3,1))
> X1 <- rbinormMH(c(0, 0), matrix(c(4, 1, 1, 4), 2, 2), sd = 0.1)
> X2 <- rbinormMH(c(0, 0), matrix(c(4, 1, 1, 4), 2, 2), sd = 2.5)
> X3 <- rbinormMH(c(0, 0), matrix(c(4, 1, 1, 4), 2, 2), sd = 10)
> plot(X1[,1], xlab = "iteration", ylab = "x1, sd = 0.1",
+      main = paste("accepted =", attr(X1, "p_accept"), "rho =",
+                   round(attr(X1, "rho1"),4)))
> cumavg1 <- cumsum(X1[,1])/1:(iterations + 1)
> lines(1:(iterations + 1), cumavg1, col = "red")
> plot(X2[,1], xlab = "iteration", ylab = "x1, sd = 2.5",
+      main = paste("accepted =", attr(X2, "p_accept"), "rho =",
+                   round(attr(X2, "rho1"),4)))
> cumavg1 <- cumsum(X2[,1])/1:(iterations + 1)
> lines(1:(iterations + 1), cumavg1, col = "red")
> plot(X3[,1], xlab = "iteration", ylab = "x1, sd = 10",
+      main = paste("accepted =", attr(X3, "p_accept"), "rho =",
+                   round(attr(X3, "rho1"),4)))
> cumavg1 <- cumsum(X3[,1])/1:(iterations + 1)
> lines(1:(iterations + 1), cumavg1, col = "red")
```



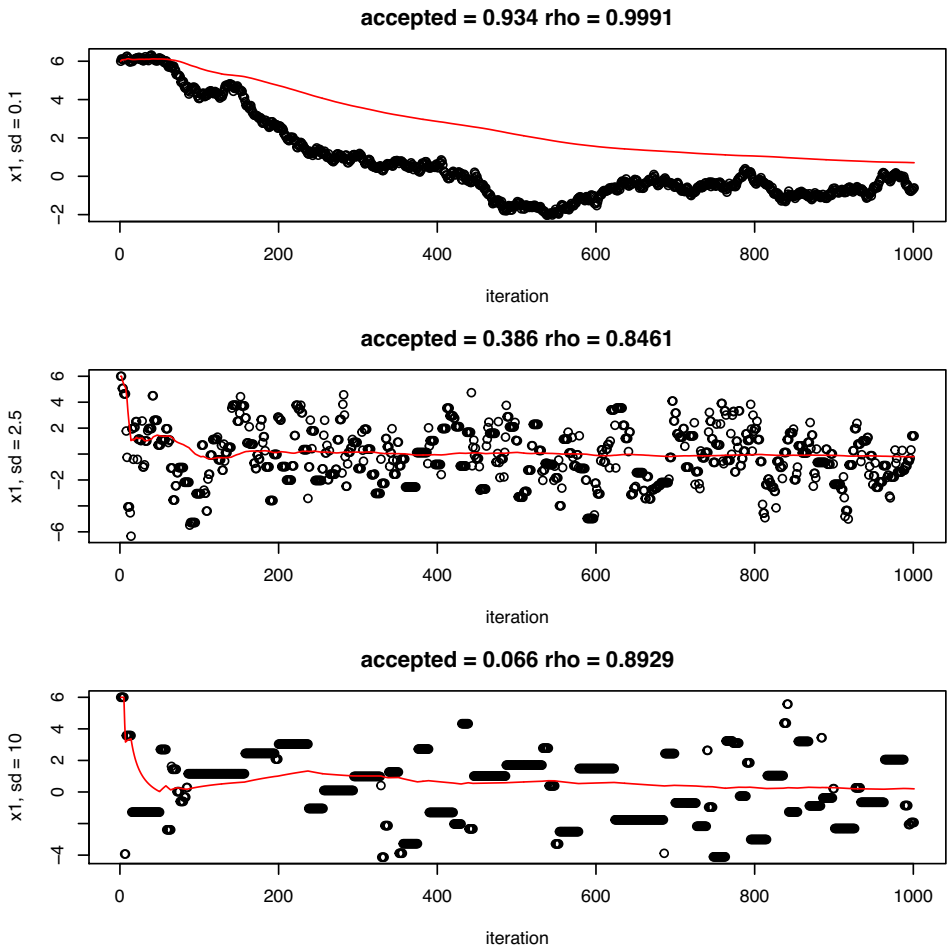
We see that when $\sigma = 0.1$ the steps of the proposal chain are too small, meaning that the MCMC chain moves too slowly, resulting in a very high autocorrelation (and correspondingly small effective sample size). Conversely when $\sigma = 10$ the MCMC chain moves slowly because so few proposal steps are accepted, again resulting in a high autocorrelation. Our original MCMC chain, with $\sigma = 2.5$, seems to be a happy medium.

(f) Now change $\mathbf{\Sigma}$ to $\begin{pmatrix} 4 & 2.8 \\ 2.8 & 4 \end{pmatrix}$ and repeat the above analysis. How do things change?

Solution:

```
> par(mfrow=c(3,1), mar=c(4,4,3,1))
> X1 <- rbinormMH(c(0, 0), matrix(c(4, 2.8, 2.8, 4), 2, 2), sd = 0.1)
> X2 <- rbinormMH(c(0, 0), matrix(c(4, 2.8, 2.8, 4), 2, 2), sd = 2.5)
> X3 <- rbinormMH(c(0, 0), matrix(c(4, 2.8, 2.8, 4), 2, 2), sd = 10)
> plot(X1[,1], xlab = "iteration", ylab = "x1, sd = 0.1",
+      main = paste("accepted =", attr(X1, "p_accept"), "rho =",
+                   round(attr(X1, "rho1"),4)))
> cumavg1 <- cumsum(X1[,1])/1:(iterations + 1)
> lines(1:(iterations + 1), cumavg1, col = "red")
> plot(X2[,1], xlab = "iteration", ylab = "x1, sd = 2.5",
+      main = paste("accepted =", attr(X2, "p_accept"), "rho =",
+                   round(attr(X2, "rho1"),4)))
> cumavg1 <- cumsum(X2[,1])/1:(iterations + 1)
> lines(1:(iterations + 1), cumavg1, col = "red")
> plot(X3[,1], xlab = "iteration", ylab = "x1, sd = 10",
+      main = paste("accepted =", attr(X3, "p_accept"), "rho =",
+                   round(attr(X3, "rho1"),4)))
> cumavg1 <- cumsum(X3[,1])/1:(iterations + 1)
```

```
> lines(1:(iterations + 1), cumavg1, col = "red")
```



Greater correlation between X_1 and X_2 means that the target chain is less like the proposal chain. This decreases the acceptance rate and hence tends to increase the autocorrelation (in the third case there is actually a tiny decrease).

2. **Variational Inference**

(a) Derive $q_1^*(x_1)$ and $q_2^*(x_2)$ and write the corresponding distribution names and their parameters.

Solution:

Let $\mathbf{\Sigma}^{-1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, where $a = \frac{\sigma_2^2}{\sigma_1^2\sigma_2^2-\sigma_{12}^2}$, $b = \frac{-\sigma_{12}}{\sigma_1^2\sigma_2^2-\sigma_{12}^2}$ and $c = \frac{\sigma_1^2}{\sigma_1^2\sigma_2^2-\sigma_{12}^2}$.

We have

$$\begin{aligned}
\log q_1(x_1) &\propto E_{x_2}[\log h(x)] \\
&\propto E_{x_2}[-\frac{1}{2}[(x_1-\mu_1)^2 a + 2(x_1-\mu_1)(x_2-\mu_2)b]] \\
&\propto -\frac{1}{2}[(x_1-\mu_1)^2 a + 2x_1 b(E_{x_2}(x_2)-\mu_2)] \\
&\propto -\frac{1}{2}[ax_1^2 - 2x_1 a[\mu_1 - \frac{b}{a}(E_{x_2}(x_2)-\mu_2)]].
\end{aligned}$$

Then we can conclude that $q_1(x_1)$ is the pdf of $N(\mu_1^*, \sigma_1^{2*})$, where

$$\begin{aligned}
\mu_1^* &= \mu_1 - \frac{b}{a}[E_{x_2}(x_2)-\mu_2] \\
&= \mu_1 + \frac{\sigma_{12}}{\sigma_2^2}[E_{x_2}(x_2)-\mu_2], \\
\sigma_1^{2*} &= \frac{1}{a} = \sigma_1^2 - \frac{\sigma_{12}^2}{\sigma_2^2}.
\end{aligned}$$

Also, we have

$$\begin{aligned}\log q_2(x_2) &\propto E_{x_1}[\log h(x)] \\ &\propto E_{x_1}[-\frac{1}{2}[(x_2-\mu_2)^2 c + 2(x_1-\mu_1)(x_2-\mu_2)b]] \\ &\propto -\frac{1}{2}[(x_2-\mu_2)^2 a + 2x_2 b(E_{x_1}(x_1)-\mu_1)] \\ &\propto -\frac{1}{2}[cx_2^2 - 2x_2 c[\mu_2 - \frac{b}{c}(E_{x_1}(x_1)-\mu_1)]].\end{aligned}$$

Then we can conclude that $q_2(x_2)$ is the pdf of $N(\mu_2^*, \sigma_2^{2*})$, where

$$\begin{aligned}\mu_2^* &= \mu_2 - \frac{b}{c}[E_{x_1}(x_1)-\mu_1] \\ &= \mu_2 + \frac{\sigma_{12}}{\sigma_1^2}[E_{x_1}(x_1)-\mu_1], \\ \sigma_2^{2*} &= \frac{1}{c} = \sigma_2^2 - \frac{\sigma_{12}^2}{\sigma_1^2}.\end{aligned}$$

And $E_{x_2}(x_2) = \mu_2^*$, $E_{x_1}(x_1) = \mu_1^*$.

(b) Derive the ELBO up to constant.

Solution:

We have

$$\begin{aligned}&E_{x_1x_2}[\log h(x) - \log q_1^*(x_1) - \log q_2^*(x_2)] \\ =&E_{x_1x_2}[\log h(x)] - E_{x_1x_2}[\log q_1^*(x_1)] - E_{x_1x_2}[\log q_2^*(x_2)].\end{aligned}$$

in which

$$\begin{aligned}E_{x_1x_2}[\log h(x)] &= E_{x_1x_2}[-\frac{1}{2}[(x_1-\mu_1)^2 a + 2(x_1-\mu_1)(x_2-\mu_2)b + (x_2-\mu_2)^2 c]] \\ &= -\frac{1}{2}[E_{x_1}[(x_1-\mu_1)^2]a + 2E_{x_1}(x_1-\mu_1)E_{x_2}(x_2-\mu_2)b + E_{x_2}[(x_2-\mu_2)^2]c],\end{aligned}$$

in which

$$\begin{aligned}E_{x_1}[(x_1-\mu_1)^2] &= E_{x_1}(x_1^2) - 2\mu_1 E_{x_1}(x_1) + \mu_1^2 \\ &= \sigma_1^{2*} + (\mu_1^*)^2 - 2\mu_1\mu_1^* + \mu_1^2, \\ E_{x_2}[(x_2-\mu_2)^2] &= \sigma_2^{2*} + (\mu_2^*)^2 - 2\mu_2\mu_2^* + \mu_2^2, \\ E_{x_1}(x_1-\mu_1) &= \mu_1^* - \mu_1, \\ E_{x_2}(x_2-\mu_2) &= \mu_2^* - \mu_2.\end{aligned}$$

And

$$\begin{aligned}E_{x_1x_2}[\log q_1^*(x_1)] &\propto E_{x_1x_2}[-\frac{1}{2}\log\sigma_1^{2*} - \frac{(x_1-\mu_1^*)^2}{2\sigma_1^{2*}}] \\ &= -\frac{1}{2}\log\sigma_1^{2*} - \frac{E_{x_1}[(x_1-\mu_1^*)^2]}{2\sigma_1^{2*}} \\ &= -\frac{1}{2}\log\sigma_1^{2*} - \frac{\sigma_1^{2*}}{2\sigma_1^{2*}} \\ &\propto -\frac{1}{2}\log\sigma_1^{2*}. \\ E_{x_1x_2}[\log q_2^*(x_2)] &\propto E_{x_1x_2}[-\frac{1}{2}\log\sigma_2^{2*} - \frac{(x_2-\mu_2^*)^2}{2\sigma_2^{2*}}] \\ &= -\frac{1}{2}\log\sigma_2^{2*} - \frac{E_{x_2}[(x_2-\mu_2^*)^2]}{2\sigma_2^{2*}} \\ &= -\frac{1}{2}\log\sigma_2^{2*} - \frac{\sigma_2^{2*}}{2\sigma_2^{2*}} \\ &\propto -\frac{1}{2}\log\sigma_2^{2*}.\end{aligned}$$

Therefore,

$$E_{x_1x_2}[\log h(x) - \log q_1^*(x_1) - \log q_2^*(x_2)]$$
$$\propto -\frac{1}{2}[E_{x_1}[(x_1-\mu_1)^2]a + 2E_{x_1}(x_1-\mu_1)E_{x_2}(x_2-\mu_2)b + E_{x_2}[(x_2-\mu_2)^2]c] + \frac{1}{2}\log\sigma_1^{2*} + \frac{1}{2}\log\sigma_2^{2*},$$

where $E_{x_1}[(x_1-\mu_1)^2], E_{x_2}[(x_2-\mu_2)^2], E_{x_1}(x_1-\mu_1), E_{x_2}(x_2-\mu_2)$ are given above.

(c) Implement the CAVI algorithm.

Solution:

```
> # mu.1 mu.2: mean for x = (x1, x2)
> # sigma2.1, sigma2.2 : diagnoal elements in Sigma
> # sigma.12 : off diagonal element in Sigma
> # initial values for mu.1*, sigma2.1*, mu.2*, sigma2.2*
> # epsilon : If the ELBO has changed by less than epsilon, the CAVI algorithm will stop
> # max.iter : maximum number of iteration
> cavi.bi.normal <- function(X, mu.1, mu.2, sigma2.1, sigma2.2, sigma.12, mu.1.vi.init,
+                sigma2.1.vi.init, mu.2.vi.init, sigma2.2.vi.init,
+                epsilon=1e-5, max.iter=100) {
+
+   mu.1.vi = mu.1.vi.init
+   sigma2.1.vi = sigma2.1.vi.init
+   mu.2.vi = mu.2.vi.init
+   sigma2.2.vi = sigma2.2.vi.init
+
+   # store the ELBO for each iteration
+   elbo = c()
+
+   # I will store mu.1*, sigma2.1*, mu.2*, sigma2.2* for each iteration
+   mu.1.vi.list = sigma2.1.vi.list = mu.2.vi.list = sigma2.2.vi.list = c()
+
+   # compute the ELBO using initial values of mu.1*, sigma2.1*, mu.2*, sigma2.2*
+   Elogq.x1 = -log(sigma2.1.vi)/2
+   Elogq.x2 = -log(sigma2.2.vi)/2
+   A = sigma2.1.vi + (mu.1.vi)^2 -2*mu.1*mu.1.vi + mu.1^2
+   B = (mu.1.vi - mu.1)*(mu.2.vi - mu.2)
+   C = sigma2.2.vi + (mu.2.vi)^2 -2*mu.2*mu.2.vi + mu.2^2
+   a = sigma2.2/(sigma2.1*sigma2.2 - sigma.12^2)
+   b = -sigma.12/(sigma2.1*sigma2.2 - sigma.12^2)
+   c = sigma2.1/(sigma2.1*sigma2.2 - sigma.12^2)
+   Elogp.x1.x2 = -0.5*(A*a + 2*b*B + C*c)
+
+   elbo = c(elbo, Elogp.x1.x2 -Elogq.x1 - Elogq.x2)
+   mu.1.vi.list = c(mu.1.vi.list, mu.1.vi)
+   sigma2.1.vi.list = c(sigma2.1.vi.list, sigma2.1.vi)
+   mu.2.vi.list = c(mu.2.vi.list, mu.2.vi)
+   sigma2.2.vi.list = c(sigma2.2.vi.list, sigma2.2.vi)
+
+   # set the change in the ELBO with 1
+   delta.elbo = 1
+
+   # number of iteration
+   n.iter = 1
+
+   # If the elbo has changed by less than epsilon, the CAVI will stop.
+   while((delta.elbo > epsilon) & (n.iter <= max.iter)){
+
+     # Update mu.1* and sigma2.1*
```

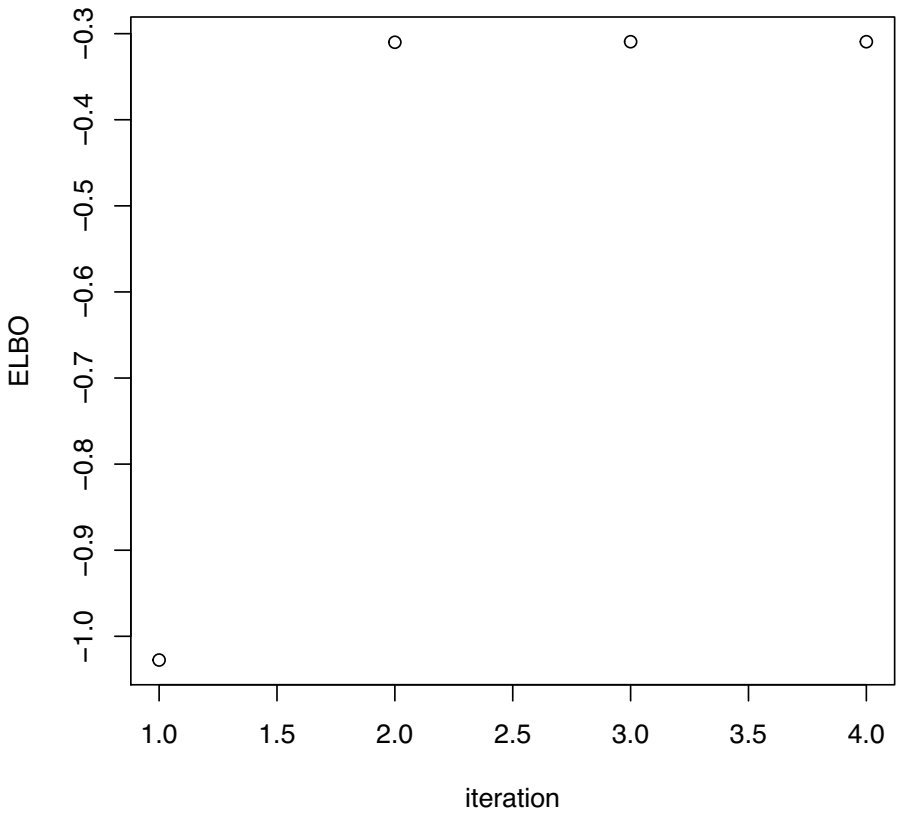
```
+     mu.1.vi = mu.1 - b/a*(mu.2.vi - mu.2)
+     sigma2.1.vi = 1/a
+
+     # Update mu.2* and sigma2.2*
+     mu.2.vi = mu.2 - b/c*(mu.1.vi - mu.1)
+     sigma2.2.vi = 1/c
+
+     # compute the ELBO using the current values of mu.1*, sigma2.1*, mu.2*, sigma2.2*
+     Elogq.x1 = -log(sigma2.1.vi)/2
+     Elogq.x2 = -log(sigma2.2.vi)/2
+     A = sigma2.1.vi + (mu.1.vi)^2 -2*mu.1*mu.1.vi + mu.1^2
+     B = (mu.1.vi - mu.1)*(mu.2.vi - mu.2)
+     C = sigma2.2.vi + (mu.2.vi)^2 -2*mu.2*mu.2.vi + mu.2^2
+     a = sigma2.2/(sigma2.1*sigma2.2 - sigma.12^2)
+     b = -sigma.12/(sigma2.1*sigma2.2 - sigma.12^2)
+     c = sigma2.1/(sigma2.1*sigma2.2 - sigma.12^2)
+     Elogp.x1.x2 = -0.5*(A*a + 2*b*B + C*c)
+
+     elbo = c(elbo, Elogp.x1.x2 -Elogq.x1 - Elogq.x2)
+     mu.1.vi.list = c(mu.1.vi.list, mu.1.vi)
+     sigma2.1.vi.list = c(sigma2.1.vi.list, sigma2.1.vi)
+     mu.2.vi.list = c(mu.2.vi.list, mu.2.vi)
+     sigma2.2.vi.list = c(sigma2.2.vi.list, sigma2.2.vi)
+
+     # compute the change in the elbo
+     delta.elbo = elbo[length(elbo)]  - elbo[length(elbo)-1]
+
+     # increase the number of iteration
+     n.iter = n.iter + 1
+   }
+
+   return(list(elbo = elbo, mu.1.vi.list = mu.1.vi.list,
+               sigma2.1.vi.list=sigma2.1.vi.list, mu.2.vi.list = mu.2.vi.list,
+               sigma2.2.vi.list=sigma2.2.vi.list))
+ }
```

(d) Obtain the approximated density $q^*(\mathbf{x}) = q_1^*(x_1)q_2^*(x_2)$ for $\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and $\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix} = \begin{pmatrix} 2 & 0.1 \\ 0.1 & 2 \end{pmatrix}$. Run the CAVI algorithm at least two times using different initial values and report $q_1^*(x_1)$ and $q_2^*(x_2)$ with the highest ELBO. For each run, check that the ELBO increase at each iteration by plotting them. Compare the approximated density $q^*(\mathbf{x}) = q_1^*(x_1)q_2^*(x_2)$ to the true density $p(\mathbf{x})$ by making contour plots.

Solution:

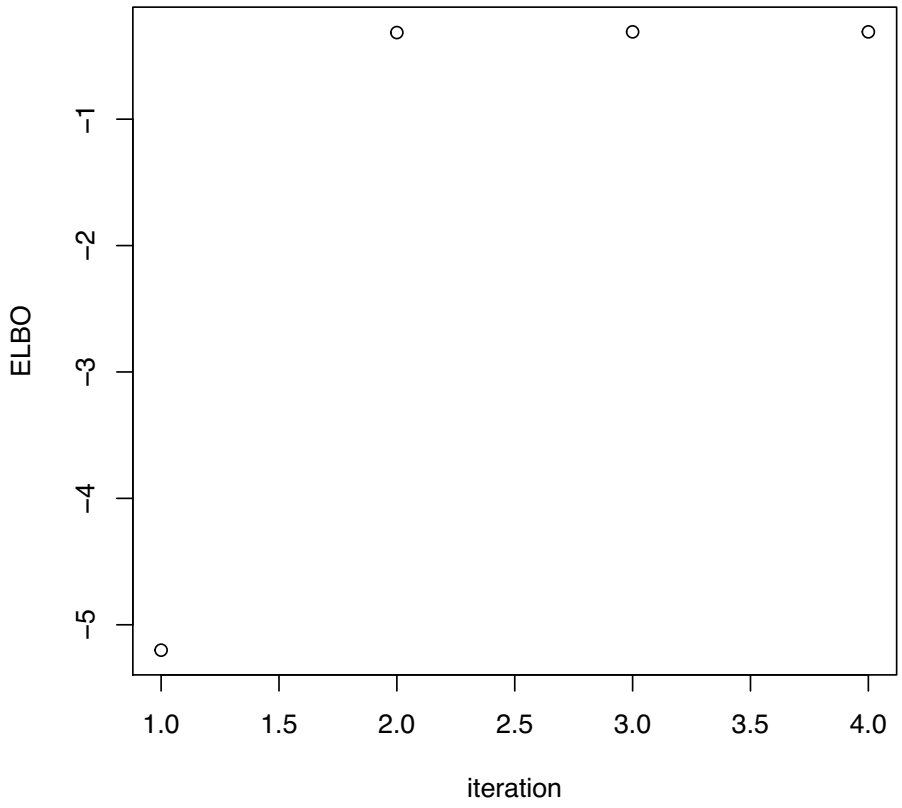
```
> mu.1 = 0
> mu.2 = 0
> sigma2.1 = 2
> sigma2.2 = 2
> sigma.12 = 0.1
> cavi1 = cavi.bi.normal(mu.1 = mu.1, mu.2 = mu.2, sigma2.1 = sigma2.1,
+       sigma2.2 = sigma2.2, sigma.12 = sigma.12, mu.1.vi.init = 1, sigma2.1.vi.init = 1,
+       mu.2.vi.init = -1, sigma2.2.vi.init = 1, epsilon=1e-5, max.iter=100)
> cavi.res = cavi1
> cavi.res$elbo

[1] -1.0275689 -0.3099809 -0.3093560 -0.3093559

> plot(cavi.res$elbo, ylab='ELBO', xlab='iteration')
> print(paste("mu.1* and sigma2.1* = (",
```

```
+             round(cavi.res$mu.1.vi.list[length(cavi.res$mu.1.vi.list)],2), ",",
+             round(cavi.res$sigma2.1.vi.list[length(cavi.res$sigma2.1.vi.list)],2),
+             ")", sep=""))
[1] "mu.1* and sigma2.1* = (0,2)"
> print(paste("mu.2* and sigma2.2* = (",
+             round(cavi.res$mu.2.vi.list[length(cavi.res$mu.2.vi.list)],2), ",",
+             round(cavi.res$sigma2.2.vi.list[length(cavi.res$sigma2.2.vi.list)],2),
+             ")", sep=""))
[1] "mu.2* and sigma2.2* = (0,2)"
```



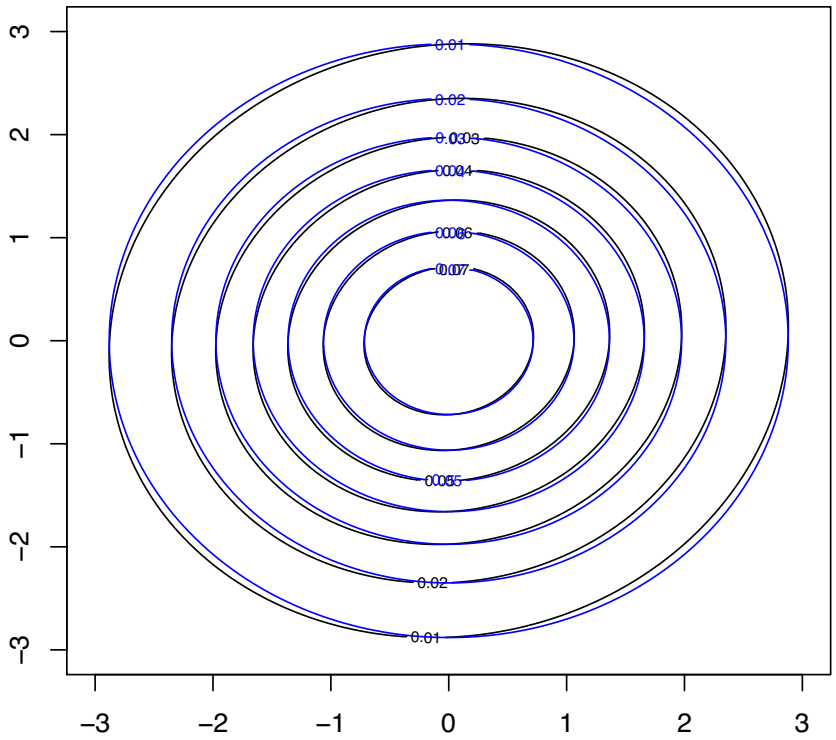

```
> cavi2 = cavi.bi.normal(mu.1 = mu.1, mu.2 = mu.2, sigma2.1 = sigma2.1,
+       sigma2.2 = sigma2.2, sigma.12 = sigma.12, mu.1.vi.init = 3, sigma2.1.vi.init = 2,
+       mu.2.vi.init = -3, sigma2.2.vi.init = 4, epsilon=1e-5, max.iter=100)
> cavi.res = cavi2
> cavi.res$elbo
[1] -5.2008807 -0.3149809 -0.3093560 -0.3093559
> plot(cavi.res$elbo, ylab='ELBO', xlab='iteration')
> print(paste("mu.1* and sigma2.1* = (",
+             round(cavi.res$mu.1.vi.list[length(cavi.res$mu.1.vi.list)],2), ",",
+             round(cavi.res$sigma2.1.vi.list[length(cavi.res$sigma2.1.vi.list)],2),
+             ")", sep=""))
[1] "mu.1* and sigma2.1* = (0,2)"
> print(paste("mu.2* and sigma2.2* = (",
+             round(cavi.res$mu.2.vi.list[length(cavi.res$mu.2.vi.list)],2), ",",
+             round(cavi.res$sigma2.2.vi.list[length(cavi.res$sigma2.2.vi.list)],2),
+             ")", sep=""))
```

```
[1] "mu.2* and sigma2.2* = (0,2)"
```



The two CAVI runs have equally highest ELBO. We can see that the approximuated densities from the runs are the same. I will use the output from the first run: $q_1^*(x_1)$ is a pdf of $\mathrm{N}(0, 2)$ and $q_2^*(x_2)$ is a pdf of $\mathrm{N}(0, 2)$. I will make the contour plot for the true density $p(\mathbf{x})$ (in black) and the contour plot for the approximated density $q^*(\mathbf{x}) = q_1^*(x_1)q_2^*(x_2)$ (in blue).

```
> library(mvtnorm)
> x.points <- seq(-3,3,length.out=100)
> y.points <- x.points
> z <- matrix(0,nrow=100,ncol=100)
> mu <- c(0,0)
> sigma <- matrix(c(2,0.1,0.1,2),nrow=2)
> for (i in 1:100) {
+   for (j in 1:100) {
+     z[i,j] <- dmvnorm(c(x.points[i],y.points[j]), mean=mu, sigma=sigma)
+   }
+ }
> contour(x.points,y.points,z)
> z <- matrix(0,nrow=100,ncol=100)
> mu <- c(0,0)
> sigma <- matrix(c(2,0,0,2),nrow=2)
> for (i in 1:100) {
+   for (j in 1:100) {
+     z[i,j] <- dmvnorm(c(x.points[i],y.points[j]), mean=mu, sigma=sigma)
+   }
+ }
> contour(x.points,y.points,z, add=TRUE, col="blue")
```



When the true correlation between x_1 and x_2 is very small, the approximated density using VI with the mean-field variational family is very similar to the true density.

(e) Obtain the approximated density $q^*(\mathbf{x}) = q_1^*(x_1)q_2^*(x_2)$ for $\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and $\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix} = \begin{pmatrix} 2 & 1.9 \\ 1.9 & 2 \end{pmatrix}$. Run the CAVI algorithm at least two times using different initial values and report $q_1^*(x_1)$ and $q_2^*(x_2)$ with the highest ELBO. For each run, check that the ELBO increase at each iteration by plotting them. Compare the approximated density $q^*(\mathbf{x}) = q_1^*(x_1)q_2^*(x_2)$ to the true density $p(\mathbf{x})$ by making contour plots.

Solution:

```
> mu.1 = 0
> mu.2 = 0
> sigma2.1 = 2
> sigma2.2 = 2
> sigma.12 = 1.9
> cavi1 = cavi.bi.normal(mu.1 = mu.1, mu.2 = mu.2, sigma2.1 = sigma2.1,
+       sigma2.2 = sigma2.2, sigma.12 = sigma.12, mu.1.vi.init = 1, sigma2.1.vi.init = 1,
+       mu.2.vi.init = -1, sigma2.2.vi.init = 1, epsilon=1e-5, max.iter=100)
> cavi.res = cavi1
> cavi.res$elbo
 [1] -15.128205  -2.860381  -2.818529  -2.784440  -2.756674  -2.734059
 [7]  -2.715639  -2.700636  -2.688415  -2.678462  -2.670355  -2.663751
[13]  -2.658373  -2.653992  -2.650424  -2.647517  -2.645150  -2.643222
[19]  -2.641652  -2.640372  -2.639331  -2.638482  -2.637791  -2.637228
[25]  -2.636769  -2.636396  -2.636092  -2.635844  -2.635642  -2.635478
[31]  -2.635344  -2.635235  -2.635146  -2.635073  -2.635014  -2.634966
```

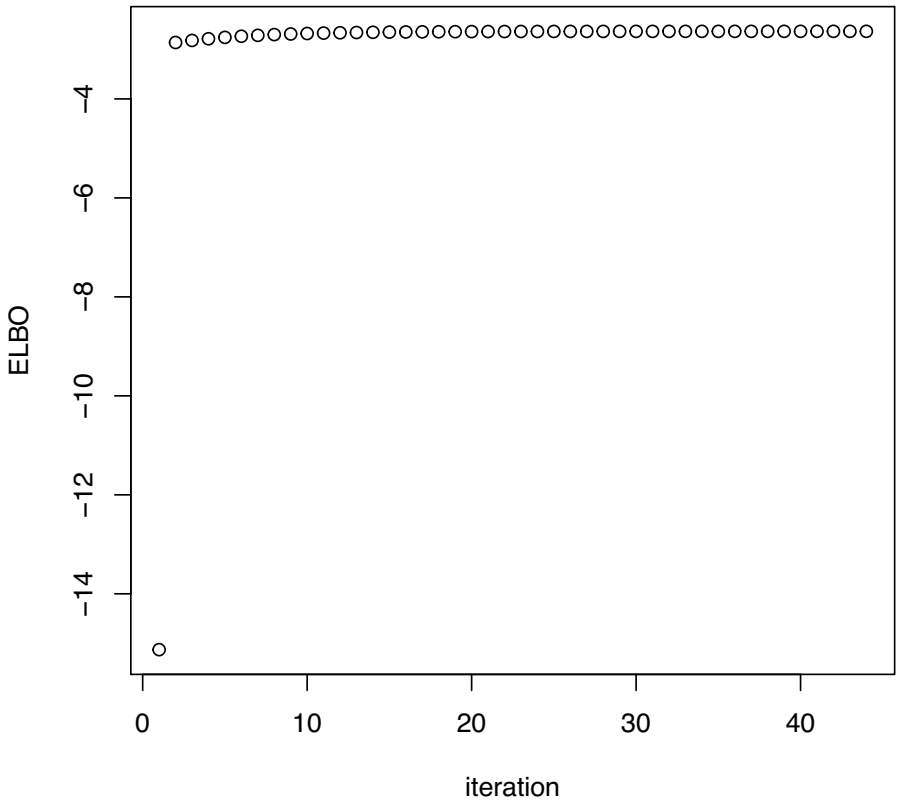
```
[37]  -2.634927  -2.634896  -2.634870  -2.634848  -2.634831  -2.634817
[43]  -2.634806  -2.634797
```

```
> plot(cavi.res$elbo, ylab='ELBO', xlab='iteration')
> print(paste("mu.1* and sigma2.1* = (",
+             round(cavi.res$mu.1.vi.list[length(cavi.res$mu.1.vi.list)],2), ",",
+             round(cavi.res$sigma2.1.vi.list[length(cavi.res$sigma2.1.vi.list)],2),
+             ")", sep=""))
```

```
[1] "mu.1* and sigma2.1* = (-0.01,0.2)"
```

```
> print(paste("mu.2* and sigma2.2* = (",
+             round(cavi.res$mu.2.vi.list[length(cavi.res$mu.2.vi.list)],2), ",",
+             round(cavi.res$sigma2.2.vi.list[length(cavi.res$sigma2.2.vi.list)],2),
+             ")", sep=""))
```

```
[1] "mu.2* and sigma2.2* = (-0.01,0.2)"
```




```
> cavi2 = cavi.bi.normal(mu.1 = mu.1, mu.2 = mu.2, sigma2.1 = sigma2.1,
+       sigma2.2 = sigma2.2, sigma.12 = sigma.12, mu.1.vi.init = 3, sigma2.1.vi.init = 2,
+       mu.2.vi.init = -3, sigma2.2.vi.init = 4, epsilon=1e-5, max.iter=100)
> cavi.res = cavi2
> cavi.res$elbo
```

```
 [1] -104.344895   -4.665381   -4.288712   -3.981914   -3.732024   -3.528488
 [7]   -3.362706   -3.227676   -3.117693   -3.028111   -2.955146   -2.895716
[13]   -2.847309   -2.807882   -2.775768   -2.749611   -2.728306   -2.710953
[19]   -2.696819   -2.685307   -2.675930   -2.668292   -2.662071   -2.657005
[25]   -2.652878   -2.649516   -2.646778   -2.644548   -2.642732   -2.641252
[31]   -2.640047   -2.639066   -2.638266   -2.637615   -2.637085   -2.636653
[37]   -2.636301   -2.636014   -2.635781   -2.635591   -2.635436   -2.635310
```

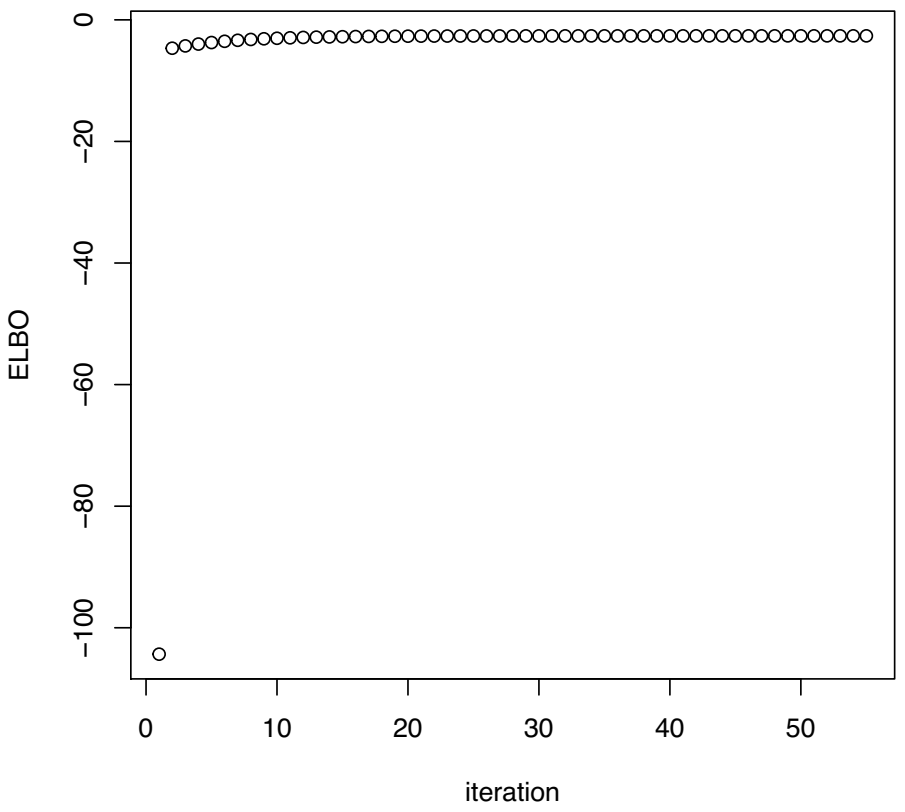
```
[43]   -2.635207   -2.635123   -2.635055   -2.634999   -2.634954   -2.634917
[49]   -2.634887   -2.634863   -2.634843   -2.634827   -2.634814   -2.634803
[55]   -2.634794
```

```
> plot(cavi.res$elbo, ylab='ELBO', xlab='iteration')
> print(paste("mu.1* and sigma2.1* = (",
+             round(cavi.res$mu.1.vi.list[length(cavi.res$mu.1.vi.list)],2), ",",
+             round(cavi.res$sigma2.1.vi.list[length(cavi.res$sigma2.1.vi.list)],2),
+             ")", sep=""))
```

```
[1] "mu.1* and sigma2.1* = (-0.01,0.2)"
```

```
> print(paste("mu.2* and sigma2.2* = (",
+             round(cavi.res$mu.2.vi.list[length(cavi.res$mu.2.vi.list)],2), ",",
+             round(cavi.res$sigma2.2.vi.list[length(cavi.res$sigma2.2.vi.list)],2),
+             ")", sep=""))
```

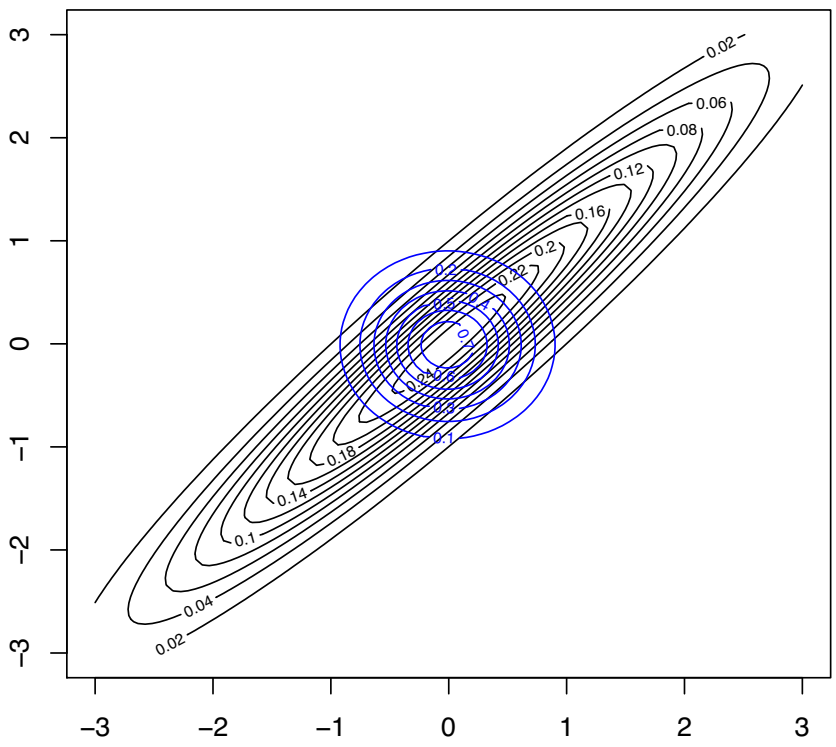
```
[1] "mu.2* and sigma2.2* = (-0.01,0.2)"
```



The two CAVI runs have equally highest ELBO. We can see that the approximuated densities from the runs are the same. I will use the output from the first run: $q_1^*(x_1)$ is a pdf of $\mathrm{N}(-0.01, 0.2)$ and $q_2^*(x_2)$ is a pdf of $\mathrm{N}(-0.01, 0.2)$. I will make the contour plot for the true density $p(\mathbf{x})$ (in black) and the contour plot for the approximated density $q^*(\mathbf{x}) = q_1^*(x_1)q_2^*(x_2)$ (in blue).

```
> library(mvtnorm)
> x.points <- seq(-3,3,length.out=100)
> y.points <- x.points
> z <- matrix(0,nrow=100,ncol=100)
> mu <- c(0,0)
> sigma <- matrix(c(2,1.9,1.9,2),nrow=2)
```

```
> for (i in 1:100) {
+   for (j in 1:100) {
+     z[i,j] <- dmvnorm(c(x.points[i],y.points[j]), mean=mu, sigma=sigma)
+   }
+ }
> contour(x.points,y.points,z)
> z <- matrix(0,nrow=100,ncol=100)
> mu <- c(-0.01,-0.01)
> sigma <- matrix(c(0.2,0,0,0.2),nrow=2)
> for (i in 1:100) {
+   for (j in 1:100) {
+     z[i,j] <- dmvnorm(c(x.points[i],y.points[j]), mean=mu, sigma=sigma)
+   }
+ }
> contour(x.points,y.points,z, add=TRUE, col="blue")
```



When the true correlation between x_1 and x_2 is very strong, the approximated density using VI with the mean-field variational family is very different from the true density. We can see that the mean is successfully captured by VI, but that the variance of $q^*(\mathbf{x})$ is controlled by the direction of smallest variance of $p(\mathbf{x})$, and that the variance along the orthogonal direction is significantly under-estimated. It is a general result that VI with the mean-field variational family tends to give approximations to the true density that are too compact.