MAST30027 Modern Applied Statistics Assignment 4

Tutorial:Wed 1-2PM, Yidi Deng

James La Fontaine

2023-10-24

Question 1

```
X = scan(file="assignment4_prob1_2023.txt", what=double())

length(X)
```

```
## [1] 100
```

```
mean(X)
```

```
## [1] 75.726
```

1a

Evaluating the parameters in the posterior distribution of the precision τ

Question 2

```r
x = scan(file="assignment4_prob2_x_2023.txt", what=double())

y = scan(file="assignment4_prob2_y_2023.txt", what=double())

length(x)
```

```
## [1] 100
```

```r
length(y)
```

```
## [1] 150
```

```r
mean(x)
```

```
## [1] 3.196441
```

```r
mean(y)
```

```
## [1] -1.979781
```

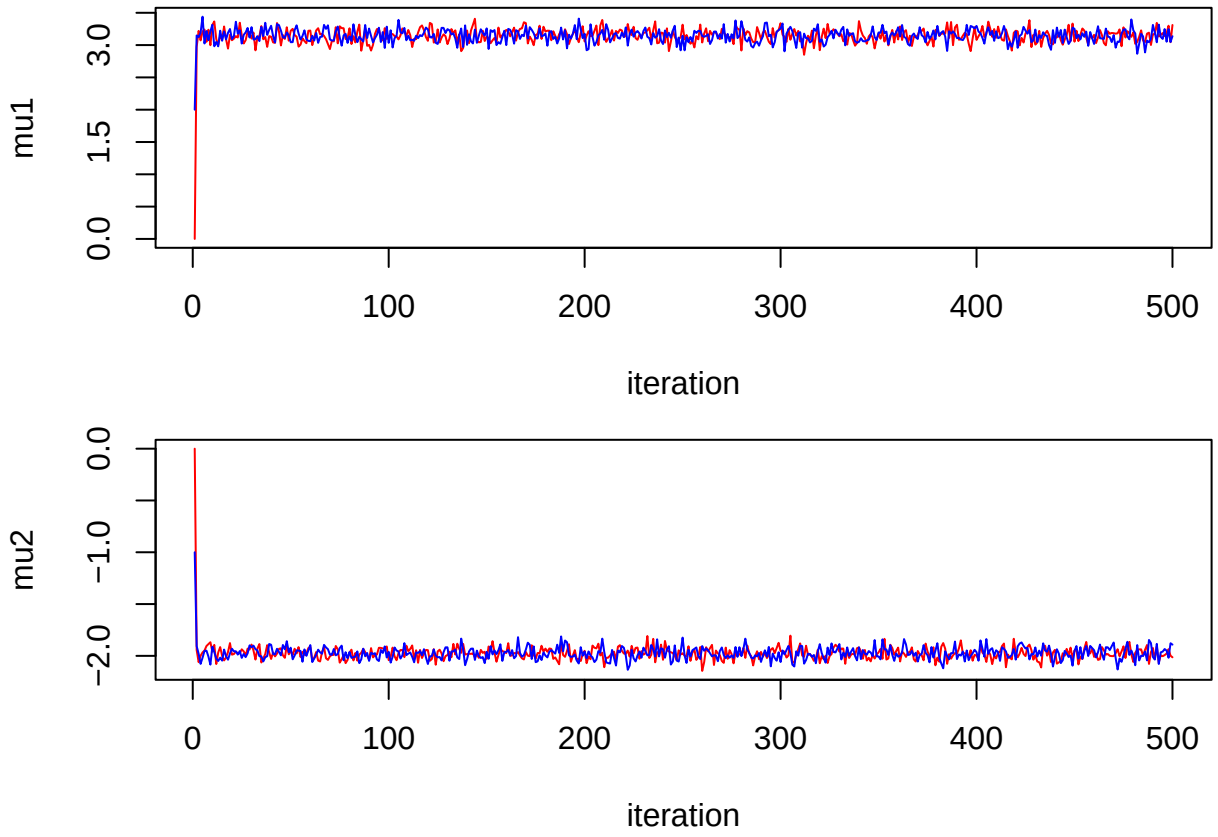
2b

```r
GibbsS <- function(mu1, mu2, nreps){

  Gsamples <- matrix(nrow=nreps, ncol=2)
  Gsamples[1,] <- c(mu1, mu2)

  # main loop
  for (i in 2:nreps) {
    mu1 <- rnorm(1, (sum(x) - 2*mu2)/103, sqrt(1/103))
    mu2 <- rnorm(1, 2*(sum(y) - mu1)/303, sqrt(1/303))
    Gsamples[i,] <- c(mu1, mu2)
  }

  return(Gsamples=Gsamples)
}
```

```r
set.seed(456)

# sample size
nreps <- 500

# initial values
mu1 <- 0
mu2 <- 0

GibbsS1 = GibbsS(mu1, mu2, nreps)
```

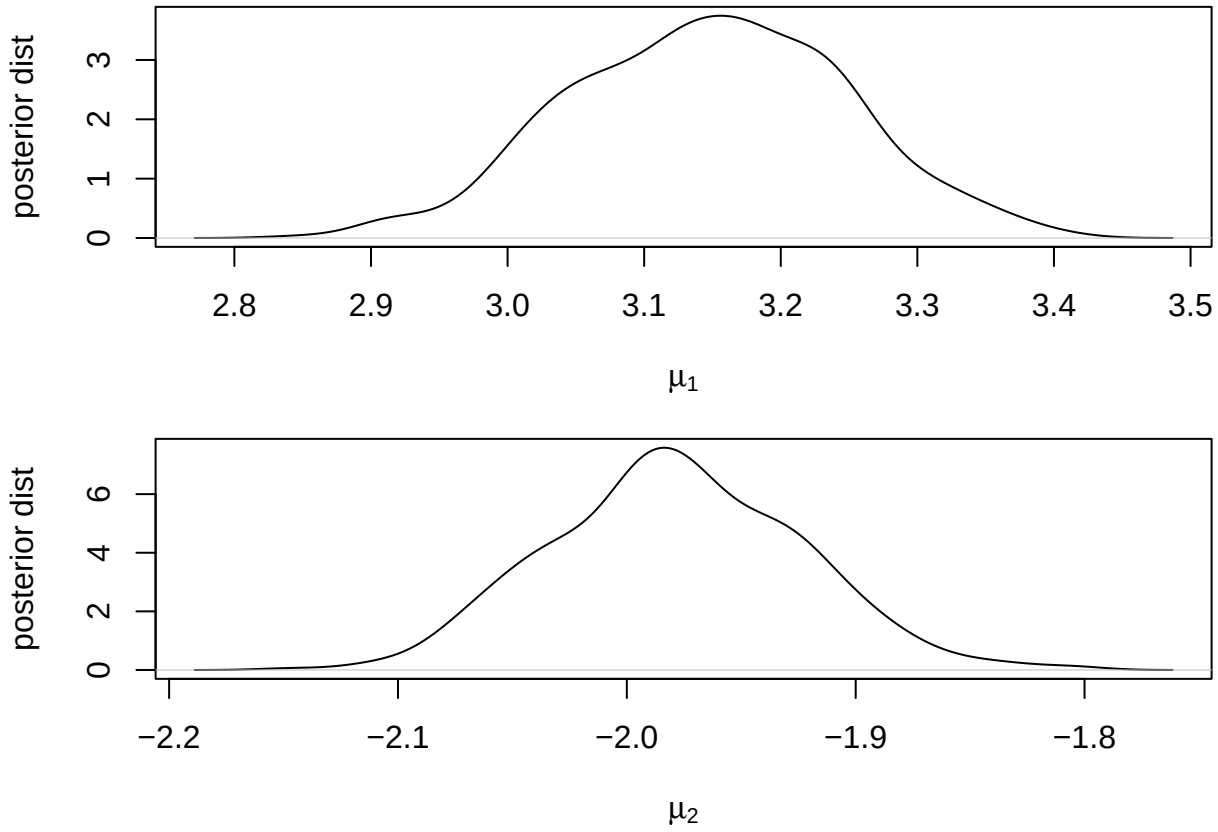
```r
# initial values
mu1 <- 2
mu2 <- -1

GibbsS2 = GibbsS(mu1, mu2, nreps)
```

```r
par(mfrow=c(2,1), mar=c(4,4,1,1))
plot(1:nreps, GibbsS1[,1], type="l", col="red", ylim = c(min(GibbsS1[,1],GibbsS2[,1]), max(GibbsS1[,1],
     xlab = "iteration", ylab ="mu1")
points(1:nreps, GibbsS2[,1], type="l", col="blue")

plot(1:nreps, GibbsS1[,2], type="l", col="red", ylim = c(min(GibbsS1[,2],GibbsS2[,2]), max(GibbsS1[,2],
     xlab = "iteration", ylab ="mu2")
points(1:nreps, GibbsS2[,2], type="l", col="blue")
```



2c

```
par(mfrow=c(2,1), mar=c(4,4,1,1))
plot(density(GibbsS1[-(1:50),1]), ylab="posterior dist", xlab=expression(mu[1]), main="")
plot(density(GibbsS1[-(1:50),2]), ylab="posterior dist", xlab=expression(mu[2]), main="")
```




```
mean(GibbsS1[-(1:50),1])
```

```
## [1] 3.146983
```

```
mean(GibbsS1[-(1:50),2])
```

```
## [1] -1.978406
```

```
quantile(GibbsS1[-(1:50),1], probs= c(0.05, 0.95))
```

```
##       5%      95%
## 2.992497 3.313051
```

```
quantile(GibbsS1[-(1:50),2], probs= c(0.05, 0.95))
```

```
##        5%       95%
## -2.065780 -1.891366
```

Estimated marginal posterior mean for μ_1 = 3.1470

Estimated marginal posterior mean for μ_2 = -1.9784

90% credible interval for μ_1 = (2.9925, 3.3131)

90% credible interval for μ_2 = (-2.0658, -1.8914)

2d

```
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))
}


######## MH algorithm ################
run_metropolis_MCMC <- function(startvalue, iterations){
    chain = array(dim = c(iterations+1,2))
    chain[1,] = startvalue
    for (i in 1:iterations){
        proposal = proposalfunction(chain[i,])

        probab = exp(posterior(proposal) - posterior(chain[i,]))
        if (runif(1) < probab){
            chain[i+1,] = proposal
        }else{
            chain[i+1,] = chain[i,]
        }
    }
    return(chain)
}

# propose new parameter values
proposalfunction <- function(param){
    return(rnorm(2,mean = param, sd= c(0.1,0.1)))
}

# evaluate log posterior at given parameter values
posterior <- function(param){
   return (likelihood(param) + prior(param))
}

# evaluate log prior at given parameter values
prior <- function(param){
    mu_vec = c(0, 0)
    sigma_matrix = matrix(c(3/5, -2/5, -2/5, 3/5), ncol=2)
    return(log(dbinorm(param, mu_vec, sigma_matrix)))
}

# evaluate log likelihood at given parameter values
likelihood <- function(param){
    mu1 = param[1]
    mu2 = param[2]
    ll_x = sum(dnorm(x, mean=mu1, sd=1, log = TRUE))
    ll_y = sum(dnorm(y, mean=mu2, sd=1/sqrt(2), log = TRUE))
    return(ll_x + ll_y)
}
```

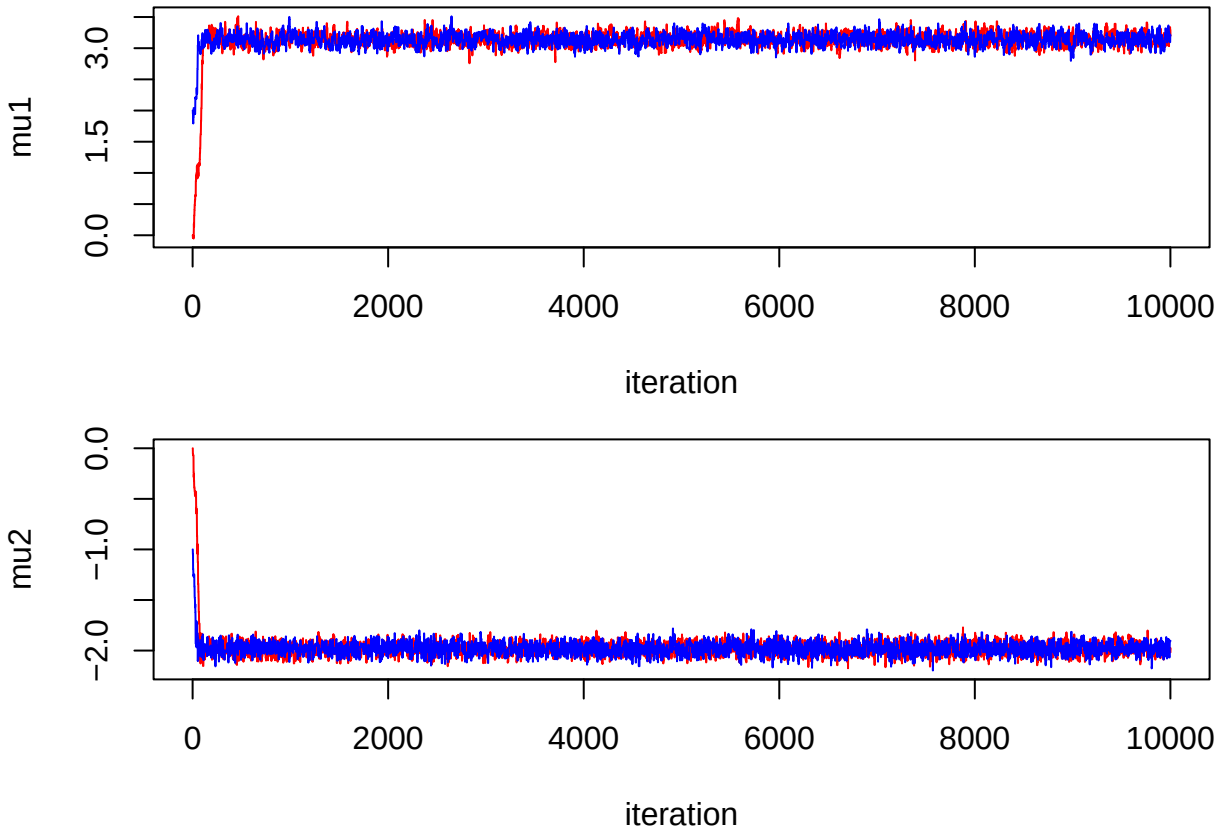
```r
set.seed(456)

nreps = 10000

# initial values
startvalue = c(0,0)
# simulate 10000 samples
MHchain1 = run_metropolis_MCMC(startvalue, nreps)

# initial values
startvalue = c(2,-1)
# simulate another 10000 samples
MHchain2 = run_metropolis_MCMC(startvalue, nreps)

# remove the first 5000 as burn-in
burnIn = 5000

# computing average acceptance probability
(acceptance1 = 1-mean(duplicated(MHchain1[-(1:burnIn),])))
```

```
## [1] 0.4373125
```

```r
(acceptance2 = 1-mean(duplicated(MHchain2[-(1:burnIn),])))
```

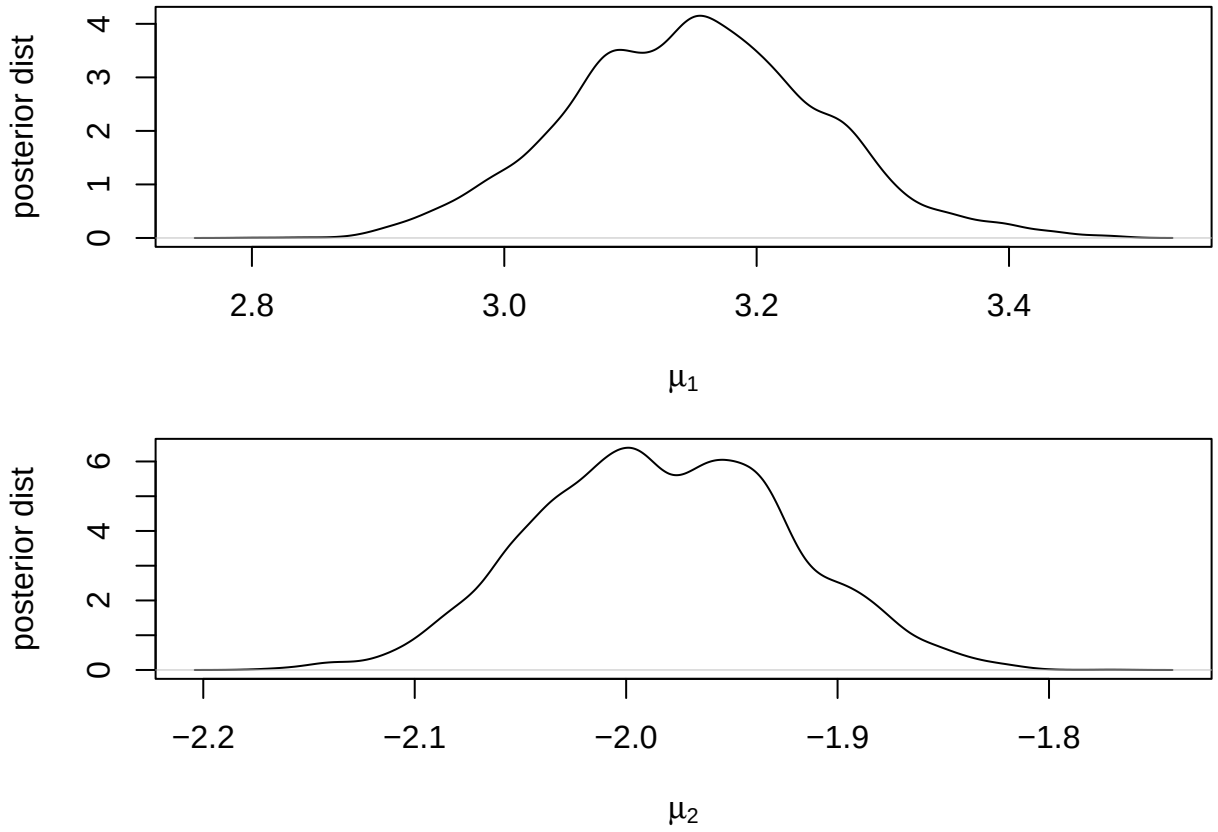
```
## [1] 0.4435113
```

```r
par(mfrow=c(2,1), mar=c(4,4,1,1))
plot(1:(nreps+1), MHchain1[,1], type="l", col="red", ylim = c(min(MHchain1[,1], MHchain2[,1]), max(MHcha
     xlab = "iteration", ylab ="mu1")
points(1:(nreps+1), MHchain2[,1], type="l", col="blue")

plot(1:(nreps+1), MHchain1[,2], type="l", col="red", ylim = c(min(MHchain1[,2], MHchain2[,2]), max(MHcha
     xlab = "iteration", ylab ="mu2")
points(1:(nreps+1), MHchain2[,2], type="l", col="blue")
```


mu1
0.0
1.5
3.0
0
2000
4000
6000
8000
10000
iteration
mu2
−2.0
−1.0
0.0
0
2000
4000
6000
8000
10000
iteration

2e

```
par(mfrow=c(2,1), mar=c(4,4,1,1))
plot(density(MHchain1[-(1:burnIn),1]), ylab="posterior dist", xlab=expression(mu[1]), main="")
plot(density(MHchain1[-(1:burnIn),2]), ylab="posterior dist", xlab=expression(mu[2]), main="")
```




```
mean(MHchain1[-(1:burnIn),1])
```

```
## [1] 3.150803
```

```
mean(MHchain1[-(1:burnIn),2])
```

```
## [1] -1.982535
```

```
quantile(MHchain1[-(1:burnIn),1], probs= c(0.05, 0.95))
```

```
##       5%      95%
## 2.988247 3.312308
```

```
quantile(MHchain1[-(1:burnIn),2], probs= c(0.05, 0.95))
```

```
##        5%       95%
## -2.077089 -1.883055
```

Estimated marginal posterior mean for μ_1 = 3.1508

Estimated marginal posterior mean for μ_2 = -1.9825

90% credible interval for μ_1 = (2.9882, 3.3123)

90% credible interval for μ_2 = (-2.0771, -1.8831)

2h

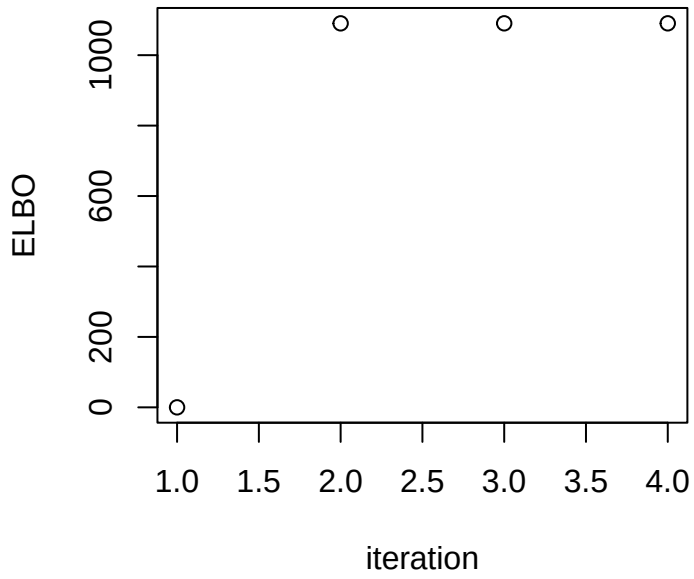
```
# x,y: data
# mu10, mu20 : prior for mu
# initial values for mu1*, mu2*,: mu1.vi.init, mu2.vi.init
# epsilon : If the ELBO has changed by less than epsilon, the CAVI algorithm will stop
# max.iter : maximum number of iteration
cavi.normal <- function(x, y, mu1.vi.init, mu2.vi.init, epsilon=1e-5, max.iter=100) {

  mu1.vi = mu1.vi.init
  mu2.vi = mu2.vi.init


  # store the ELBO for each iteration
  elbo = c()

  # I will store mu*, sigma2*, a*, b* for each iteration
  mu1.vi.list = mu2.vi.list = c()

  # compute the ELBO using initial values of mu*

  elbo = c(elbo, (-103/2) * mu1.vi^2 + mu1.vi*sum(x) - (303/2) * mu2.vi^2 + 2*mu2.vi*sum(y) - 2*mu1.vi*m
  mu1.vi.list = c(mu1.vi.list, mu1.vi)
  mu2.vi.list = c(mu2.vi.list, mu2.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.vi and sigma2.vi
    mu1.vi = (sum(x) - 2*mu2.vi)/103
    mu2.vi = (2*sum(y)-2*mu1.vi)/303


    # compute the ELBO using the current values of mu*
    elbo = c(elbo, (-103/2) * mu1.vi^2 + mu1.vi*sum(x) - (303/2) * mu2.vi^2 + 2*mu2.vi*sum(y) - 2*mu1.vi
    mu1.vi.list = c(mu1.vi.list, mu1.vi)
    mu2.vi.list = c(mu2.vi.list, mu2.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, mu1.vi.list = mu1.vi.list,
              mu2.vi.list=mu2.vi.list))
```

```
}
```

```
set.seed(456)

cavi1 = cavi.normal(x, y, mu1.vi.init=0, mu2.vi.init=0, epsilon=1e-5, max.iter=100)
cavi.res = cavi1
cavi.res$elbo
```

```
## [1]    0.000 1090.321 1090.397 1090.397
```

```
plot(cavi.res$elbo, ylab='ELBO', xlab='iteration')
```



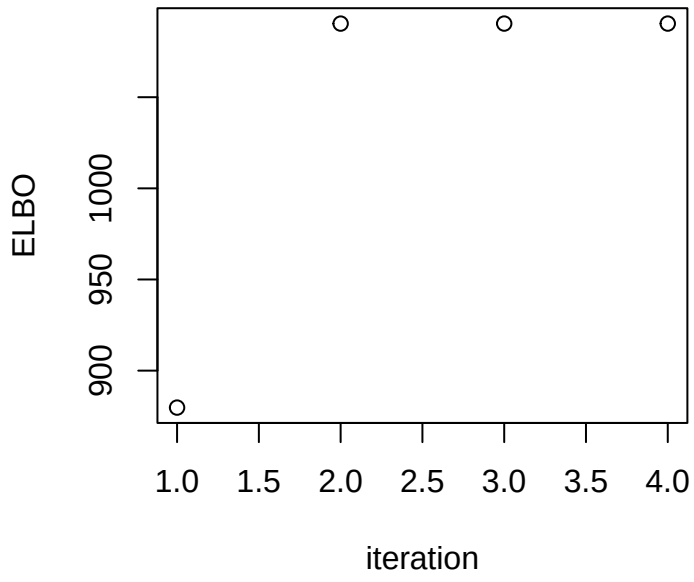

```
print(paste("mu1* and mu2* = (",
            round(cavi.res$mu1.vi.list[length(cavi.res$mu1.vi.list)],2), ",",
            round(cavi.res$mu2.vi.list[length(cavi.res$mu2.vi.list)],2), ")", sep=""))
```

```
## [1] "mu1* and mu2* = (3.14,-1.98)"
```

```
cavi2 = cavi.normal(x, y, mu1.vi.init=2, mu2.vi.init=-1, epsilon=1e-5, max.iter=100)
cavi.res = cavi2
cavi.res$elbo
```

```
## [1]  879.7223 1090.3781 1090.3968 1090.3968
```

```
plot(cavi.res$elbo, ylab='ELBO', xlab='iteration')
```




```
print(paste("mu1* and mu2* = (",
            round(cavi.res$mu1.vi.list[length(cavi.res$mu1.vi.list)],2), ",",
            round(cavi.res$mu2.vi.list[length(cavi.res$mu2.vi.list)],2), ")", sep=""))
```

```
## [1] "mu1* and mu2* = (3.14,-1.98)"
```