Lecture 4.11 example 3

preliminaries

```
#clear workspace
rm(list=ls())
#initalize random seed
set.seed(15354)
```

specify simulation parameters

```
beta0 = 0
beta1 = 0.5
beta2 = - 0.5
sigma = 0.5

alpha = 0.5

nObs = 200

nPredictPerX = 100

stepBeta0Grid = 0.1
stepBeta1Grid = 0.05
stepBeta2Grid = 0.05
stepSigmaGrid = 0.1
```

build model objects

```
beta0Grid = seq(-1,1, by = stepBeta0Grid)
beta1Grid = seq(-1,1, by = stepBeta1Grid)
beta2Grid = seq(-1,1, by = stepBeta2Grid)
sigmaGrid = seq(stepSigmaGrid,2, by = stepSigmaGrid)

# grid utility functions
stepSize = function(grid) {
  if (length(grid)==1) {
    step = 1
  }
  else {
    step = (max(grid) - min(grid)) / (length(grid) - 1)
  }
  return(step)
}

# build priors
buildPriorMultivar = function(beta0Grid,beta1Grid,beta2Grid,sigmaGrid) {
  # build useful grid objects
```

```
  nBeta0Grid = length(beta0Grid)
  nBeta1Grid = length(beta1Grid)
  nBeta2Grid = length(beta2Grid)
  nSigmaGrid = length(sigmaGrid)
  #
  prior = array( rep(1, nBeta0Grid * nBeta1Grid * nBeta2Grid * nSigmaGrid ),
                 dim = c(nBeta0Grid, nBeta1Grid, nBeta2Grid, nSigmaGrid ))
  #
  for (nB0 in 1:nBeta0Grid) {
    for (nB1 in 1:nBeta1Grid) {
      for (nB2 in 1:nBeta2Grid) {
        for (nSig in 1:nSigmaGrid) {
          # change next expression to set different priors
          prior[nB0,nB1,nB2, nSig] = 1 / nSig^2
        }
      }
    }
  }
  return(prior)
}

buildPriorUnivar = function(beta0Grid,betaPGrid,sigmaGrid) {
  # build useful grid objects
  nBeta0Grid = length(beta0Grid)
  nBetaPGrid = length(betaPGrid)
  nSigmaGrid = length(sigmaGrid)
  #
  prior = array( rep(1, nBeta0Grid * nBetaPGrid * nSigmaGrid ),
                 dim = c(nBeta0Grid, nBetaPGrid, nSigmaGrid ))
  #
  for (nB0 in 1:nBeta0Grid) {
    for (nBP in 1:nBetaPGrid) {
      for (nSig in 1:nSigmaGrid) {
        # change next expression to set different priors
        prior[nB0,nBP, nSig] = 1 / nSig^2
      }
    }
  }
  return(prior)
}

#likelihood
likelihoodMultivar = function(y,x1, x2, b0L, b1L, b2L, sL){
  loglike = sum(log(dnorm(y-b0L-b1L*x1-b2L*x2, mean = 0, sd=sL)))
  like = exp(loglike)
  return(like)
}

likelihoodUnivar = function(y,xP, b0L, bPL, sL){
  loglike = sum(log(dnorm(y-b0L-bPL*xP, mean = 0, sd=sL)))
  like = exp(loglike)
  return(like)
}
```

```r
#compute posterior function
compPostMultivar = function(y,x1, x2, prior, beta0Grid,beta1Grid,beta2Grid,sigmaGrid) {
  # build useful grid objects
  nBeta0Grid = length(beta0Grid)
  nBeta1Grid = length(beta1Grid)
  nBeta2Grid = length(beta2Grid)
  nSigmaGrid = length(sigmaGrid)
  #initialize local posterior
  post = array( rep(-1, nBeta0Grid * nBeta1Grid * nBeta2Grid * nSigmaGrid ),
                dim = c(nBeta0Grid, nBeta1Grid, nBeta2Grid, nSigmaGrid ))
  # compute posterior
  for (nBeta0 in 1:nBeta0Grid) {
    b0 = beta0Grid[nBeta0]
    #print(paste("bo = ", b0))
    for (nBeta1 in 1:nBeta1Grid) {
      b1 = beta1Grid[nBeta1]
      for (nBeta2 in 1:nBeta2Grid) {
        b2 = beta2Grid[nBeta2]
        for (nSigma in 1:nSigmaGrid) {
          s = sigmaGrid[nSigma]
          post[nBeta0,nBeta1,nBeta2,nSigma] = likelihoodMultivar(y,x1,x2,b0,b1,b2,s) * prior[nBeta0,nBe
        }
      }
    }
  }
  # normalize posterior
  post = post / ( sum(post) *stepSize(beta0Grid) * stepSize(beta1Grid) * stepSize(beta2Grid) * stepSize
  # return
  return(post)
}


#compute posterior function
compPostUnivar = function(y,xP, prior, beta0Grid,betaPGrid,sigmaGrid) {
  # build useful grid objects
  nBeta0Grid = length(beta0Grid)
  nBetaPGrid = length(betaPGrid)
  nSigmaGrid = length(sigmaGrid)
  #initialize local posterior
  post = array( rep(-1, nBeta0Grid * nBetaPGrid * nSigmaGrid ),
                dim = c(nBeta0Grid, nBetaPGrid, nSigmaGrid ))
  # compute posterior
  for (nBeta0 in 1:nBeta0Grid) {
    b0 = beta0Grid[nBeta0]
    #print(paste("bo = ", b0))
    for (nBetaP in 1:nBetaPGrid) {
      bP = betaPGrid[nBetaP]
      for (nSigma in 1:nSigmaGrid) {
        s = sigmaGrid[nSigma]
        post[nBeta0,nBetaP,nSigma] = likelihoodUnivar(y,xP,b0,bP,s) * prior[nBeta0,nBetaP,nSigma]
      }
    }
```

```
  }
  # normalize posterior
  post = post / ( sum(post) * stepSize(beta0Grid) * stepSize(betaPGrid) * stepSize(sigmaGrid))
  # return
  return(post)
}
```

simulate dataset

```
z = rnorm(nObs, 0,2)
x1 = alpha * z + (1-alpha) * rnorm(nObs, 0,2)
x2 = alpha * z + (1-alpha) * rnorm(nObs, 0,2)
y = rnorm(nObs, beta0 + beta1 * x1 + beta2 * x2, sigma )
```

fit models

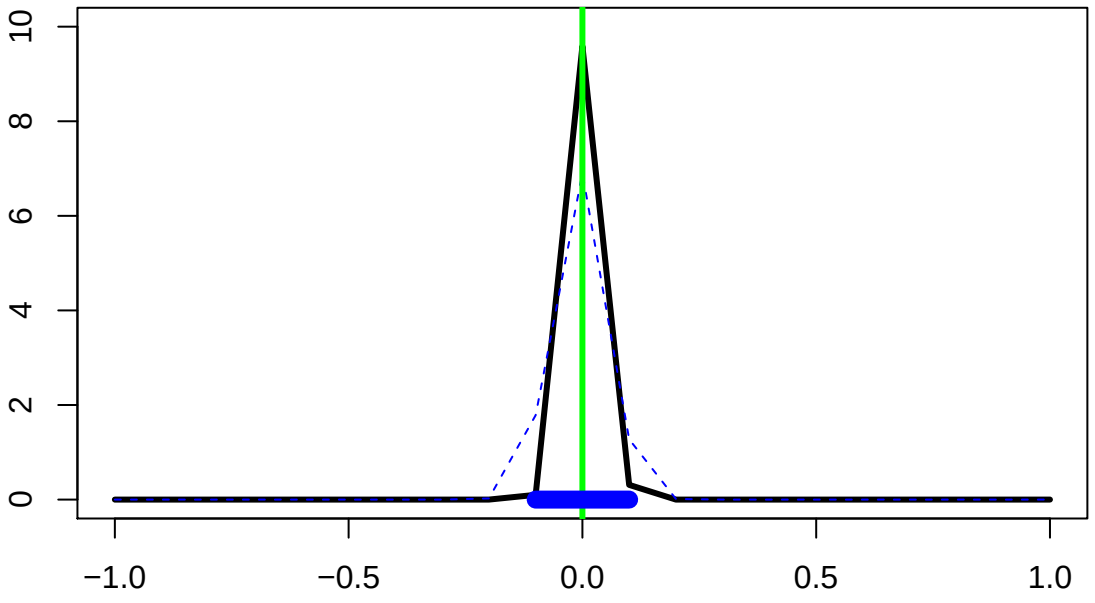
```
## model 1: y ~ bo + b1 * x1

# build priors
priorM1 = buildPriorUnivar(beta0Grid,beta1Grid,sigmaGrid)

#compute posterior function iteratively using batches of 100 observations
for (k in 1:floor(nObs/100)) {
  #print(k)
  #print('++++++++++++++++++++')
  y_batch = y[(1+(k-1)*100):(k*100)]
  x1_batch = x1[(1+(k-1)*100):(k*100)]
  postM1 = compPostUnivar(y_batch,x1_batch,
                          priorM1, beta0Grid,beta1Grid,sigmaGrid)
  priorM1 = postM1
}


#compute marginal posteriors
margPostBeta0M1 = apply(postM1,c(1),sum)
margPostBeta0M1 = margPostBeta0M1 / ( sum(margPostBeta0M1) *stepSize(beta0Grid))

margPostBeta1M1 = apply(postM1,c(2),sum)
margPostBeta1M1 = margPostBeta1M1 / (sum(margPostBeta1M1) * stepSize(beta1Grid))

margPostSigmaM1 = apply(postM1,c(3),sum)
margPostSigmaM1 = margPostSigmaM1 / (sum(margPostSigmaM1) * stepSize(sigmaGrid))
```

```
# full model: y ~ bo + b1 * x1 + b2 * x2

# build priors
priorM3 = buildPriorMultivar(beta0Grid,beta1Grid,beta2Grid,sigmaGrid)

#compute posterior function iteratively using batches of 100 observations
for (k in 1:floor(nObs/100)) {
  #print(paste("k = ", k))
```

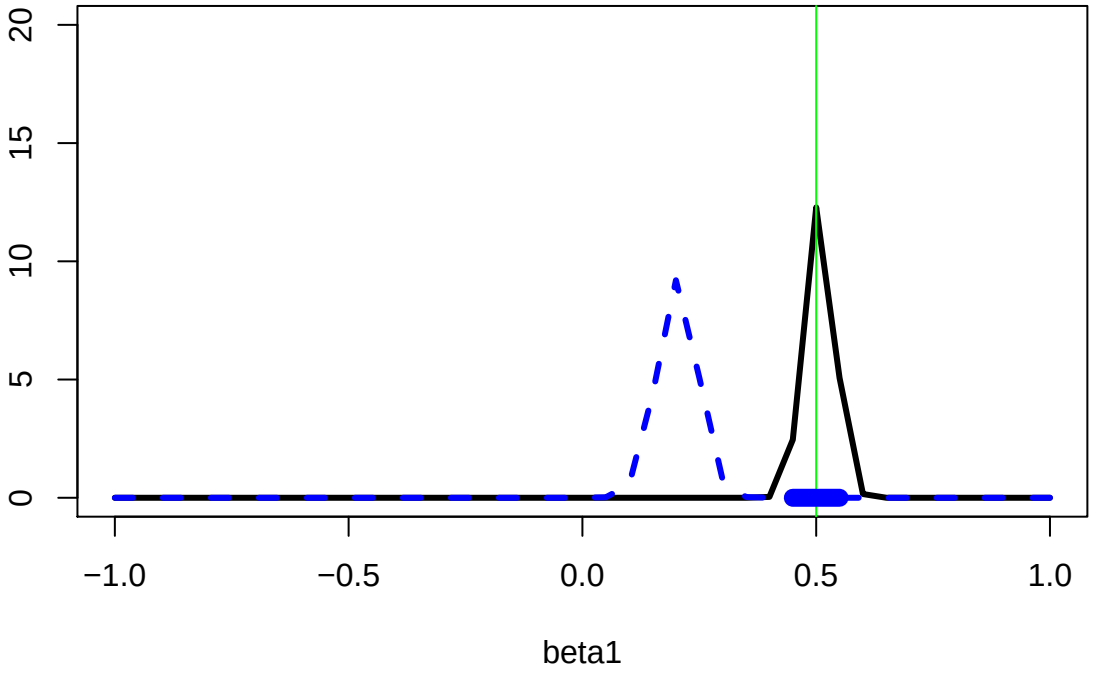
```
  #print('++++++++++++++++++++')
  y_batch = y[(1+(k-1)*100):(k*100)]
  x1_batch = x1[(1+(k-1)*100):(k*100)]
  x2_batch = x2[(1+(k-1)*100):(k*100)]
  postM3 = compPostMultivar(y_batch,x1_batch,x2_batch,
                            priorM3, beta0Grid,beta1Grid,beta2Grid,sigmaGrid)
  priorM3 = postM3
}

#compute marginal posteriors
margPostBeta0M3 = apply(postM3,c(1),sum)
margPostBeta0M3 = margPostBeta0M3 / (sum(margPostBeta0M3) * stepSize(beta0Grid))

margPostBeta1M3 = apply(postM3,c(2),sum)
margPostBeta1M3 = margPostBeta1M3 / (sum(margPostBeta1M3) * stepSize(beta1Grid))

margPostBeta2M3 = apply(postM3,c(3),sum)
margPostBeta2M3 = margPostBeta2M3 / (sum(margPostBeta2M3) * stepSize(beta2Grid))

margPostSigmaM3 = apply(postM3,c(4),sum)
margPostSigmaM3 = margPostSigmaM3 / (sum(margPostSigmaM3) * stepSize(sigmaGrid))
```

compare marginal posteriors across models

```
plot(beta0Grid, margPostBeta0M3,
     xlab = "beta0", ylab="",
     type = "l", lwd = 3,
     ylim=c(0,10))
points(beta0Grid, margPostBeta0M1,
     type = "l", lwd = 1, col = "blue", lty=2)
abline(v=beta0, lwd=3, col="green")
segments(beta0-stepSize(beta0Grid), 0,beta0+stepSize(beta0Grid),0, lwd=9, col="blue" )
```




```
plot(beta1Grid, margPostBeta1M3,
     xlab = "beta1", ylab="",
     type = "l", lwd = 3,
     ylim=c(0,20))
points(beta1Grid, margPostBeta1M1,
       type = "l", lwd = 3, col = "blue", lty=2)
abline(v=beta1, lwd=1, col="green")
segments(beta1-stepSize(beta1Grid), 0,beta1+stepSize(beta1Grid),0, lwd=9, col="blue" )
```




```
plot(sigmaGrid, margPostSigmaM3,
     xlab = "sigma", ylab="",
```

```
     type = "l", lwd = 3)
points(sigmaGrid, margPostSigmaM1,
       type = "l", lwd = 1, col = "blue", lty=2)
abline(v=sigma, lwd=2, col="green")
segments(sigma-stepSize(sigmaGrid), 0,sigma+stepSize(sigmaGrid),0, lwd=9, col="blue" )
```