Homework 9 Responses

Zeyi Han, Shubhi Sharma, Margaret Swift

2020-04-09

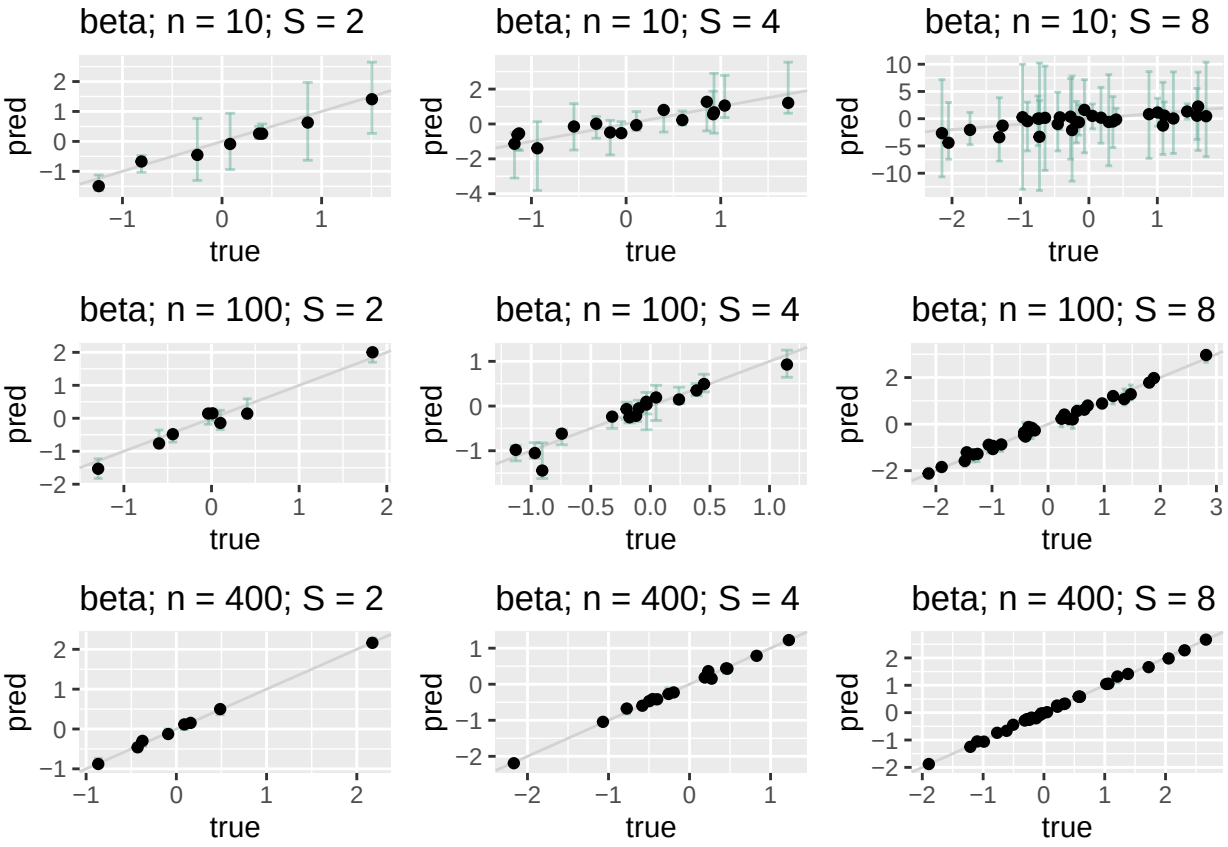
Exercise 1

Conduct simulation experiments to determine how sample size and number of response variables affects estimates of the coefficients and covariance matrix.

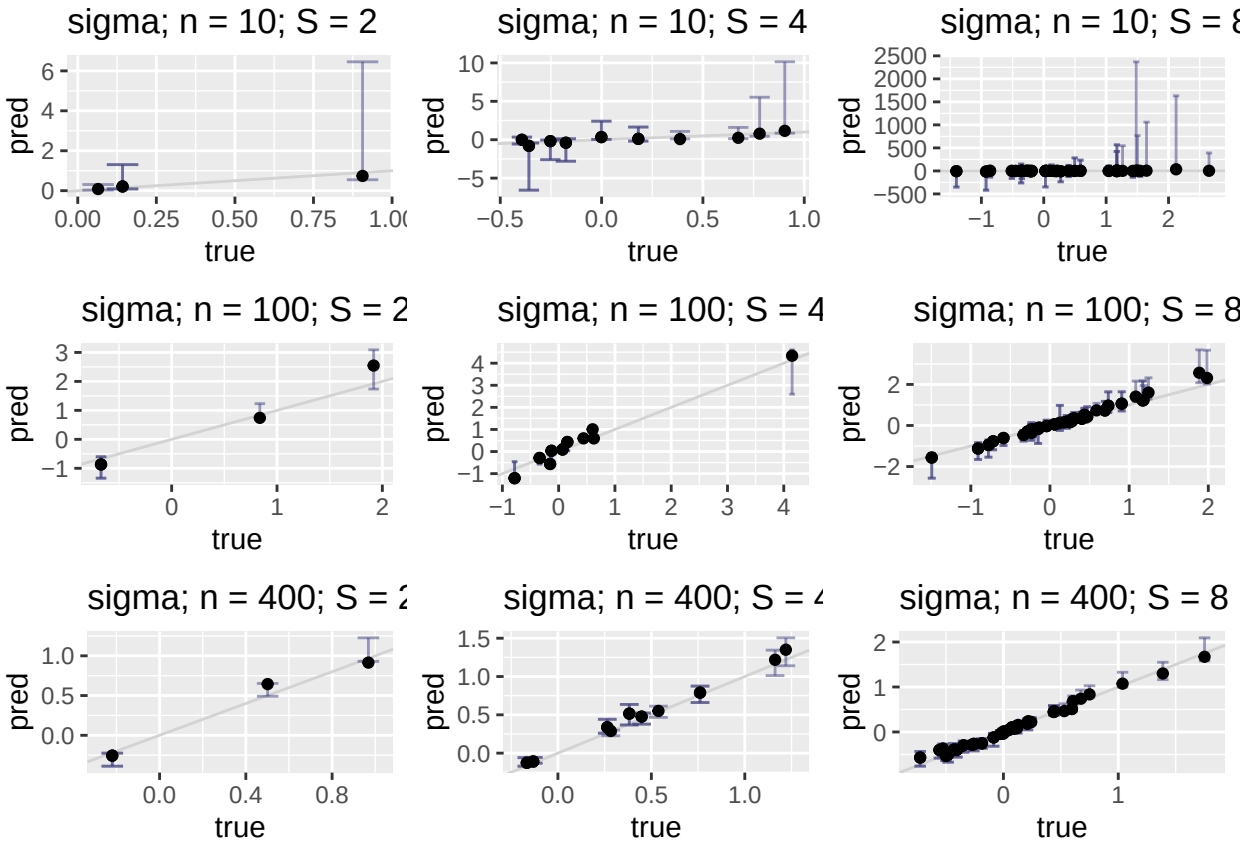
Taking a look at the first row of the beta graphs, you can see that the credible intervals are much wider as you increase the number of species (from left to right); this is less visible for the other `n` value rows but still present. It seems like the opposite is happening for the y predictions; as the number of species increases, the predicted values get closer to the true values.

Conversely, increasing `n` (going down one column) makes estimates more accurate. This also makes sense, as increasing the sample size will allow the sampler to make better predictions. Note how the error bars decrease from the first to the second rows for the `beta` and `Sigma` plots. I don't see as much of an affect for the predicted y-values, other than increasing the number of points.

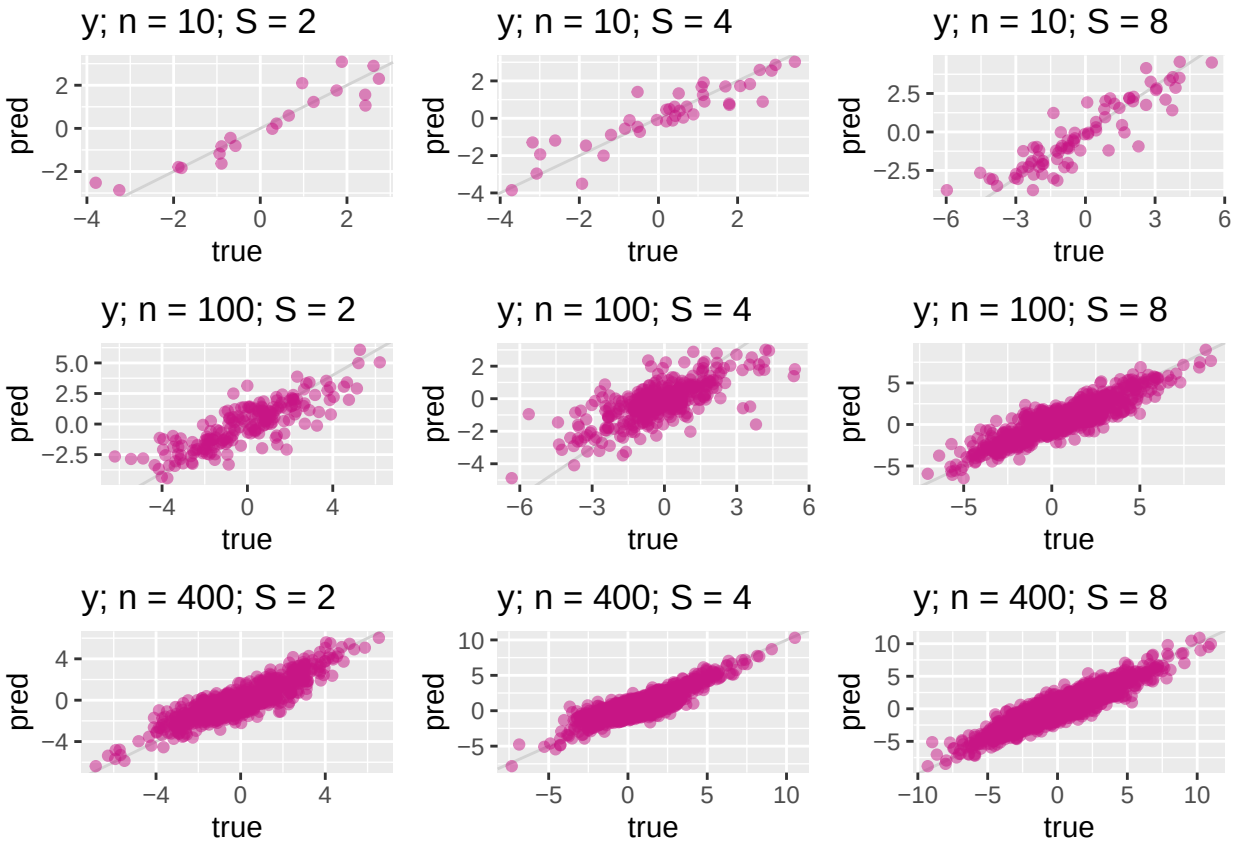
Predicted coefficients vs. observed:



Predicted covariances vs. observed:



Predicted responses vs. observed:



Exercise 2

Jon has a survey with one binary response (`typeNames = 'PA'`) and three ordinal responses (ordinal counts, `'OC'`). Simulate his data using `gjamSimData` and analyze it using `gjam`. Using the documentation for `gjam`:

a) What are the partition estimates and what role do they play?

b) Which are better predicted by the fitted model, the `PA` data or the `OC` data? Why might they differ?

Exercise 3

Select 10 species from the BBS data and do a model selection (using DIC) to evaluate a limited number of variable combinations.

a) How do the DIC values compare with overall sensitivity?

b) With inverse prediction?

Because it took so long (read: hours) to run all of these models, we captured the DIC values and hardcoded them into this table after the first run:

Table 1: Model with DIC values for different GJAM models

Model Name	Formula	DIC
outBBS	~ winterTemp + therm + def + ncld	10539378
outBBS2	~ obsTemp + prec + ncld + soil	10566072
outBBS3	~ Longitude + Latitude + ncld	10591289
outBB4	~ def + prec + obsTemp + ncld	10634372

From the analysis of DIC for different combinations of GJAM models for a given 20 species, we find that the first model with covariates `winterTemp`, `therm`, `def` and `ncld` has the lowest DIC value and therefore is preferred to other models with higher DICs.

This model has extremely good inverse prediction and covariate sensitivity for ncld is ~ 0.02, therm $= 0.02$, wintertemp < 0.05 and finally winterTemp ~ 0.2.

Compared to other models with higher DICs, while they also have good inverse predictions, the model sensitivites are usually much more uncertain (i.e. larger credible intervals). Notably, most models have similar sensitivities.

Exercise 4

Using the block of code that generates the predictive coefficient of variation, provide an interpretation of what you see.

I had trouble getting this code to run properly, but this is what I would expect to see:

Looking at the section of code below, I see it is looping over each of the values in `ii`, which holds a random sample of 4 predicted y means held in the list `yPredMu`. `zj` is the predicted yvalue's standard error divided by its mean with some error term equal to 0.01. We then run a custom Clark Function, `values2contour()`, which takes the latlon values from the prediction `predGrid` and creates a contour map with height of contours given by the scaled standard error previously calculated, `zj`.

After creating the contour map, the border of the region of interest is overlayed on top (`maps::map` line). We then recover the inputted `y` data from the GJAM output `out`, scale it by 1000 hours, and then again by its maximum value in order to get reasonable scales for our plot point areas. Finally, this chunk plots each site's latitude and longitude on top of the contour map, using the scaled values in `ydata` as the radius (and, likely, color, although I don't know what .getColor does precisely) for the various data points.

The final output, then, should be a contour map of sampled predicted mean y values, overlaid with the observed data points, scaled and colored by the magnitude of observed response variable.

Appendix

```
#-------------------------------------------------------------------------
# LOADING
knitr::opts_chunk$set(echo=FALSE, eval=TRUE, cache=FALSE)
pacman::p_load(ggplot2, gjam, gridExtra, RColorBrewer, reshape2)
source('../clarkfunctions2020.R')
Rcpp::sourceCpp('../cppFns.cpp')
# any libraries and data here

#-------------------------------------------------------------------------
# FUNCTIONS
# put functions here

#-------------------------------------------------------------------------
## EXERCISE 1

##########################################################################
# Simulation
##########################################################################
mycol <- c("#69b3a2", "#404080", "#C71585")
n <- c(10, 100, 400)
S <- c(2, 4, 8)
iter.df <- data.frame(n=rep(n,each=length(S)) , S=rep(S, by=length(n)))
```

```r
glist <- list()
for (i in 1:nrow(iter.df)) {
  # Parameters
  n <- iter.df$n[i]  # sample size
  S <- iter.df$S[i]    # responses
  Q <- 4    # predictors

  # Coefficient matrix
  beta <- matrix( rnorm(Q), Q )
  for (j in 2:S) { beta <- cbind(beta, matrix( rnorm(Q), Q )) }

  # Design matrix
  x <- matrix( rnorm(n*Q), n, Q)
  x[, 1] <- 1

  # Covariance matrix
  Sigma <- cov( .rMVN(5, 0, diag(S) ) )

  # Labeling things
  xnames <- paste0('x', 1:Q)
  ynames <- paste0('y', 1:S)
  colnames(beta) <- rownames(Sigma) <- colnames(Sigma) <- ynames
  rownames(beta) <- colnames(x) <- xnames

  # Simulating y
  y  <- x%*%beta + .rMVN(n, 0, Sigma)

  ##############################################################################
  # Recovering covariates
  ##############################################################################
  bg <- beta*0
  sg <- Sigma*0

  rownames(bg) <- colnames(x) <- xnames
  colnames(bg) <- rownames(sg) <- colnames(sg) <- ynames

  ng <- 2000                        # setup Gibbs sampler
  bgibbs <- matrix(0,ng,S*Q)
  sgibbs <- matrix(0,ng,S*S)
  predy  <- matrix(0,ng,S*n)        # predict the data

  colnames(bgibbs) <- as.vector( outer(xnames,ynames,paste,sep='_') )
  colnames(sgibbs) <- as.vector( outer(ynames,ynames,paste,sep='_') )

  IXX <- solve(crossprod(x))       # only do this once
  df  <- n - Q + S - 1             # Wishart

  for(g in 1:ng){

    bg <- .updateBetaMVN(x,y,sg)
    sg <- suppressMessages(.updateWishart(x, y, df, beta=bg, IXX=IXX)$sigma)

    sgibbs[g,] <- sg
```

```
    bgibbs[g,] <- bg

    predy[g,] <- as.vector( .rMVN(n,x%*%bg,sg) )  # predicted y
  }

  rownames(bg) <- colnames(x) <- xnames
  colnames(bg) <- rownames(sg) <- colnames(sg) <- ynames

  bmu <- colMeans(bgibbs)
  bci <- apply(bgibbs, 2, quantile, c(.025,.975))
  sci <- apply(sgibbs, 2, quantile, c(.025,.975))

  ypred <- matrix( colMeans(predy),n,S )
  bg.df <- data.frame(melt(beta), melt(bg)$value, lower=bci[1,], upper=bci[2,])
  sg.df <- data.frame(melt(Sigma), melt(sg)$value, lower=sci[1,], upper=sci[2,])
  names(bg.df) <- names(sg.df) <- c('x', 'y', 'true', 'pred', 'lower', 'upper')
  y.df <- data.frame(true=melt(y)$value, pred=melt(ypred)$value)

  p1 <- ggplot(bg.df, aes(true, pred)) +
    geom_abline(slope=1, intercept=0, color='lightgrey') +
    geom_errorbar(aes(ymin=lower, ymax=upper), alpha=0.5, color=mycol[1], width=.1) +
    geom_point() + ggtitle(paste0('beta; n = ', n, '; S = ', S))
  p2 <- ggplot(sg.df, aes(true, pred)) +
    geom_abline(slope=1, intercept=0, color='lightgrey') +
    geom_errorbar(aes(ymin=lower, ymax=upper), alpha=0.5, color=mycol[2], width=.1) +
    geom_point() + ggtitle(paste0('sigma; n = ', n, '; S = ', S))
  p3 <- ggplot(y.df, aes(true, pred)) +
    geom_abline(slope=1, intercept=0, color='lightgrey') +
    geom_point(alpha=0.5, color=mycol[3]) +
    ggtitle(paste0('y; n = ', n, '; S = ', S))

  glist[[i]] <- list(n=n, S=S, bgplot=p1, sgplot=p2, ypredplot=p3)
}
# would make this a loop but couldn't figure & ran out of time...
grid.arrange(glist[[1]]$bgplot, glist[[2]]$bgplot, glist[[3]]$bgplot,
             glist[[4]]$bgplot, glist[[5]]$bgplot, glist[[6]]$bgplot,
             glist[[7]]$bgplot, glist[[8]]$bgplot, glist[[9]]$bgplot)
grid.arrange(glist[[1]]$sgplot, glist[[2]]$sgplot, glist[[3]]$sgplot,
             glist[[4]]$sgplot, glist[[5]]$sgplot, glist[[6]]$sgplot,
             glist[[7]]$sgplot, glist[[8]]$sgplot, glist[[9]]$sgplot)
grid.arrange(glist[[1]]$ypredplot, glist[[2]]$ypredplot, glist[[3]]$ypredplot,
             glist[[4]]$ypredplot, glist[[5]]$ypredplot, glist[[6]]$ypredplot,
             glist[[7]]$ypredplot, glist[[8]]$ypredplot, glist[[9]]$ypredplot)
#------------------------------------------------------------------------------
## EXERCISE 2

#------------------------------------------------------------------------------
## EXERCISE 3

tmp <- gjamTrimY(ydata, 1000)
ytrim <- tmp$y                  #trimmed version of ydata
dim(ytrim)
```

```
reductList <- list(N = 20, r = 15)
modelList  <- list(ng = 2000, burnin = 500, typeNames='DA',
                   reductList = reductList)
formula1 <- as.formula(~ winterTemp + therm + def + nlcd)

outBBS  <- gjam(formula1, xdata, ydata = ytrim, modelList = modelList)
outBBS$fit$DIC

formula2 <- as.formula(~ obsTemp + prec + ncld + soil)
outBBS2  <- gjam(formula1, xdata, ydata = ytrim, modelList = modelList)
outBBS2$fit$DIC

formula3 <- as.formula(~ Longitude + Latitude + ncld )
outBBS3 <- gjam(formula1, xdata, ydata = ytrim, modelList = modelList)
outBBS3$fit$DIC

formula4 <- as.formula(~ def + prec + obsTemp + ncld)
outBB4  <- gjam(formula1, xdata, ydata = ytrim, modelList = modelList)
outBB4$fit$DIC

#-------------------------------------------------------------------------
## EXERCISE 4

# coefficient of variation
for(j in 1:4){
  i <- ii[j]
  wi <- which(colnames(yPredMu) == i)
  zj <- yPredSe[,i]/(yPredMu[,i] + .01)
  values2contour(xx=predGrid[,'lon'],yy=predGrid[,'lat'],
                 z=zj,nx=ngrid,ny=ngrid,col=colm,
                 zlevs=slevs,lwd=.1,add=F,fill=T)

  maps::map(boundary=T,col='grey',lwd=2,xlim=mapx,ylim=mapy,add=T)

  ydat <- out$inputs$y[,wi]*1000  # per 1000 hrs effort
  ydat <- ydat/max(ydat,na.rm=T)
  colj <- .getColor('darkblue',ydat)
  symbols(xdata$Longitude, xdata$Latitude, circles=ydat, inches=F, add=T,
          fg = colj, bg = colj)
  title(i)
}
```