Week 1 Lab: Getting started in RJAGS

## Part 1: Getting JAGS installed and running

WinBUGS is the language used for most of the textbooks, including the ones we will be using in this class. However, WinBUGS is not being actively developed and its not available for Mac. For these reasons, in this course we will use JAGS (Just Another Gibbs Sampler) and interface with it through the R package ‘rjags’. Fortunately, all of these programs share a common syntax which only minor changes; moving between different programs is relatively straightforward.

If you haven’t already, download JAGS for your operating system from here:

<http://sourceforge.net/projects/mcmc-jags/files/JAGS/3.x/>

Then download ‘rjags’ from RStudio, as you would any other package. Rjags should find your installation of jags, but if it doesn’t, then you’ll have to figure out how to point rjags to your JAGS installation. (This will differ by platform, and I have no better advice than to Google it, which you all can do as well as I can.)

You’ll need the ‘mcmc’ package as well, so you might as well download that while we’re at it.

\*Before we get very far with Bayesian computation, a caveat. Bayesian computation is infuriating, idiosyncratic, and while it works brilliantly most of the time, it fails disastrously the rest of the time. Development in this area is rapid, so code that worked one week may not work next week, and different versions of jags may run differently. (Usually, the code will fail to run in one version but work in another version.) Code running on different platform may run differently. JAGS, like its fellows BUGS programs WinBUGS and OpenBUGS, crashes regularly. Unlike WinBUGS, which responds to a crash by popping open a meaningless page of wingdings (which apparently identify the error, but not in ay way you could figure out), JAGS usually supplies some information about the source of the error and is, in this way, slightly less infuriating to use than WinBUGS. This is all to say that while I can teach you the basic principles of Bayesian statistics, and teach you the basic elements of doing Bayesian analyses in JAGS, I may not be able to debug your code, especially if the error is computational (something that has gone wrong with the sampler inside JAGS), and your colleagues may also be unable to help. You may find some discussion of the bug online, and that’s where I would suggest looking. I mention this at the outset because you should go into this process prepared for confusing JAGS malfunctions, and be comforted by the fact that myself and my colleagues run into similar agonizing problems and emerge victorious (eventually) with code that works.

We will convince ourselves that everything works by running a simple test code to find the mean of a normal distribution. We’ll simulate some data and then run JAGS to find the mean. I have uploaded the code to Bb, which can be run just like any other script. If it works, than you can be fairly certain everything is installed properly. Don’t worry about what it all means just yet.

## Part 2: Playing around with joint distributions

First, a warm up. Let’s say you have two independent variables X and Y with the following probability distributions

and

Question #1: What is the joint probability distribution of (X,Y)?

Questions #2: Show that .

Next we’ll work through some more complex cases, all in the name of trying to give you some intuition on how these multivariate distributions work.

In each of the following three cases, find the (1) joint distribution P(X,Y), the two marginal distributions P(X) and P(Y) (2 and 3), and (4) determine if X and Y are independent.

Case #1: (X,Y) are uniformly distributed on the square

Case #2: (X,Y) are uniformly distributed on the triangle

Case #3: (X,Y) are uniformly distributed on the circle

Once you have worked this out analytically, we’ll try sampling from these distributions, and plotting the marginal probabilities. How you might sample from these distributions?

Exercise #1: Write a function to sample from these three joint distributions, and draw 1000 values from the joint distribution. Plot the values, and put the marginal histograms on the X and Y axes. There are many ways to include the marginal histograms. I did a quick google search and found some code that would work (which uses the R package ‘gridextra’). There are probably better way; I’ll be interested to see what you all come up with.

hist\_top <- ggplot()+geom\_histogram(aes(rnorm(100)))

empty <- ggplot()+geom\_point(aes(1,1), colour="white")+ opts(axis.ticks=theme\_blank(), panel.background=theme\_blank(), axis.text.x=theme\_blank(),

axis.text.y=theme\_blank(), axis.title.x=theme\_blank(),

axis.title.y=theme\_blank())

scatter <- ggplot()+geom\_point(aes(rnorm(100), rnorm(100)))

hist\_right <- ggplot()+geom\_histogram(aes(rnorm(100)))+coord\_flip()

grid.arrange(hist\_top, empty, scatter, hist\_right, ncol=2, nrow=2, widths=c(4, 1), heights=c(1, 4))

Here is another set of code I found, which puts density plots (not actual histograms) at the margins

#placeholder plot - prints nothing at all

empty <- ggplot()+geom\_point(aes(1,1), colour="white") +

theme(

plot.background = element\_blank(),

panel.grid.major = element\_blank(),

panel.grid.minor = element\_blank(),

panel.border = element\_blank(),

panel.background = element\_blank(),

axis.title.x = element\_blank(),

axis.title.y = element\_blank(),

axis.text.x = element\_blank(),

axis.text.y = element\_blank(),

axis.ticks = element\_blank()

) #scatterplot of x and y variables

scatter <- ggplot(xy,aes(xvar, yvar)) +

geom\_point(aes(color=zvar)) +

scale\_color\_manual(values = c("orange", "purple")) + theme(legend.position=c(1,1),legend.justification=c(1,1))

#marginal density of x - plot on top

plot\_top <- ggplot(xy, aes(xvar, fill=zvar)) + geom\_density(alpha=.5) +

scale\_fill\_manual(values = c("orange", "purple")) + theme(legend.position = "none")

#marginal density of y - plot on the right

plot\_right <- ggplot(xy, aes(yvar, fill=zvar)) + geom\_density(alpha=.5) +

coord\_flip() +

scale\_fill\_manual(values = c("orange", "purple")) + theme(legend.position = "none")

#arrange the plots together, with appropriate height and width for each row and column

grid.arrange(plot\_top, empty, scatter, plot\_right, ncol=2, nrow=2, widths=c(4, 1), heights=c(1, 4))