JAGS tutorial with replication of Joseph, et al.

Nathaniel Hendrix

April 9, 2018

Objective and Motivation

My goal for this document is to teach the basic principles of conducting a Bayesian analysis of diagnostic data in JAGS ("Just Another Gibbs Sampler"). I am assuming a basic knowledge of Bayesian ideas, such as the definition of a prior, posterior, and likelihood. I will also not be explaining the concept of a Gibbs sampler (for that I recommend John Kruschke's islands example). Rather, I'll focus on creating a user-friendly document showing how to implement these ideas.

This is similar material to what is covered by the article, "A tutorial in estimating the prevalence of disease in humans and animals in the absence of a gold standard" by Fraser Lewis and Paul Gorgerson – and this write-up borrows from their work quite a bit. Since this article's publication in 2012, though, several tools have come out that make the interface between R and JAGS more seamless. Therefore, while they spread their model, analysis, and priors across several files, we're able to do everything within a single file.

For my example, I'll be performing a replication of a well-known paper by Lawrence Joseph, Theresa Gyorkos, and Louis Coupal (henceforth JGC) entitled "Bayesian estimation of disease prevalence and parameters of diagnostic tests in the absence of a gold standard." Their example concerns two diagnostic tests for the parasite *Strongyloides* that were administered to Cambodian immigrants to Canada in the early 1980s. As the title suggests, neither one of these is a gold standard test. In fact, they give quite different pictures of the prevalence of this parasite: 24.7% of patients tested positive on the stool test, while 77.2% tested positive on the serology test. The goal of the paper is therefore to estimate the parasite's true prevalence and to estimate the performance of the two tests.

Required packages

JAGS is a stand-alone package that must be downloaded and installed outside of the R environment before installing the R packages from CRAN.

Once JAGS has been downloaded, you'll need to install two more packages, `rjags` and `coda`. The former is to run the JAGS model from R, while the latter focuses on display of the results.

```
require(rjags)
```

```
## Loading required package: rjags

## Loading required package: coda

## Linked to JAGS 4.2.0

## Loaded modules: basemod,bugs
```

```
require(coda)
```

Simulating the data

JAGS depends on individual observations of data. Therefore, we'll need to create a dataset for it to work its way through.

JGC write that, of 162 participants, 40 tested positive for *Strongyloides* in the stool examination and 125 tested positive in the serology test. Of those testing positive for the stool test, 38 tested positive in the serology test; of those testing negative for the stool test, 87 tested positive in the serology test.

```
stool    <- c(rep(1, 40), rep(0, 122))
serology <- c(rep(1, 38), rep(0, 2), rep(1, 87), rep(0, 35))
N        <- 162
set.seed(108)
```

The one-test model

Before diving into the JAGS code, it's worth taking a look at the model that JGC propose for establishing the prevalence and test characteristics when a single test is used. This will then form the basis of the model that uses data from both tests.

One of the difficulties that the problem posed by JGC contains is that our estimate of the prevalence is dependent upon the performance of the test, just as our understanding of the performance of the test is dependent on our knowledge of the prevalence of the disease in this population. JGC get around this difficulty by recruiting tropical disease experts to create priors for the test characteristics.

On the other hand, they had literally no idea about the prevalence of this parasite in the population at hand, so they used a uniform prior ranging from 0 to 1.

The way that a JAGS model works is that it iterates through each observation and updates the priors with each run. You might notice, too, that the models are written upside-down – the priors come *after* the model itself. Using p for prevalence, y for the observed test result, and JGC's terminology of S for sensitivity and C for specificity, our two one-test models (with values taken from Table 1 of JGC) are:

```
# create the model for stool exam
model_string_stool = "
model {
  for(i in 1:N){
    y[i] ~ dbern(S*p+(1-C)*(1-p))
  }
  S ~ dbeta(4.44, 13.31)
  C ~ dbeta(71.25, 3.75)
  p ~ dunif(0,1)
}
"

# create the model for serology
model_string_sero = "
model {
  for(i in 1:N){
    y[i] ~ dbern(S*p+(1-C)*(1-p))
  }
  S ~ dbeta(21.96, 5.49)
  C ~ dbeta(4.1, 1.76)
  p ~ dunif(0, 1)
}
"
```

Our posterior (y) and each prior (S, C, and p) all require that a distribution is specified. The three distributions in our model are Bernoulli for the outcome, beta for the prior of the test characteristics, and uniform for the prior of the prevalence.

The next step will be to write these models to temporary text files.

```
writeLines(model_string_stool, con = "stool_model.txt")
writeLines(model_string_sero, con  = "sero_model.txt")
```

After this, we'll be good to initialize our model. We'll allow for 1,000 adaptation runs and 10,000 burn-in runs before we start recording the results of our sampler.

```
jags_stool <- jags.model("stool_model.txt",
                         data     = list('y' = stool,
                                         'N' = N),
                         n.chains = 4,
                         n.adapt  = 1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 162
##    Unobserved stochastic nodes: 3
##    Total graph size: 501
##
## Initializing model
```

```
update(jags_stool, 10000) #burn-in
```

The coda package comes in handy once we're ready to record the results of the chains that will form the basis for our analysis. All the runs we've done to this point have been for adaptation purposes or have been unlikely to represent the final conclusions of the analysis. The package provides handy data types for storing, manipulating, and understanding the results of your sampler.

Below, I'm running 10,000 samples and displaying summary statistics about the results, which we can compare to JGC.

```
stool_samples <- coda.samples(jags_stool,
                              c("p", "S", "C"),
                              n.iter = 10000)
summary(stool_samples)
```

```
##
## Iterations = 11001:21000
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##     Mean      SD  Naive SE Time-series SE
## C 0.9469 0.02691 0.0001346      0.0002163
## S 0.3136 0.06921 0.0003461      0.0009064
## p 0.7392 0.16663 0.0008331      0.0021778
##
## 2. Quantiles for each variable:
##
##     2.5%    25%    50%    75%  97.5%
## C 0.8831 0.9318 0.9515 0.9668 0.9859
## S 0.2092 0.2643 0.3019 0.3517 0.4798
## p 0.3936 0.6182 0.7571 0.8775 0.9877
```

Let's see how these results compare to those found in Table 5 of JGC. For the prevalence, they found a median of 0.74 with 95% credible interval of 0.41-0.98. We're a little off on the low end, but not bad! For sensitivity, they found 0.30 (0.21-0.47) and here we're extremely close. Same for specificity, where they found 0.95 (0.88-0.99).

I'll repeat the same process below for the serology test.

```
jags_sero <- jags.model("sero_model.txt",
                         data     = list('y' = serology,
                                         'N' = N),
                         n.chains = 4,
                         n.adapt  = 1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 162
##    Unobserved stochastic nodes: 3
##    Total graph size: 501
##
## Initializing model
```

```
update(jags_sero, 10000) #burn-in

sero_samples <- coda.samples(jags_sero,
                             c("p", "S", "C"),
                             n.iter = 10000)
summary(sero_samples)
```

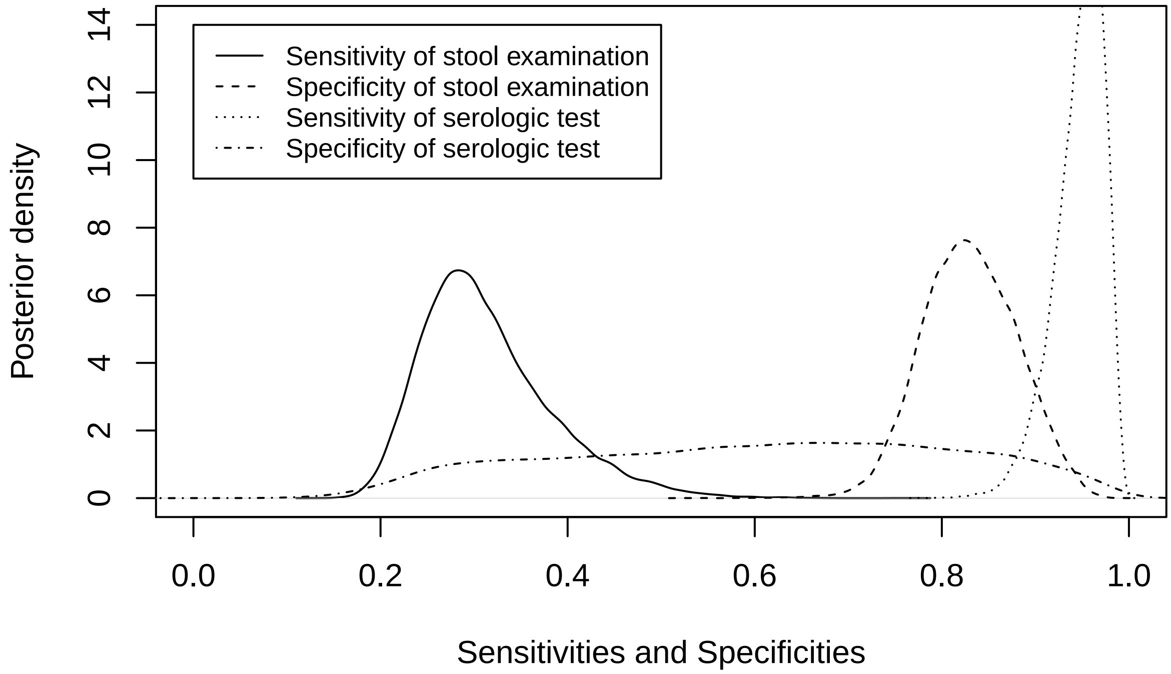
```
##
## Iterations = 11001:21000
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##     Mean      SD  Naive SE Time-series SE
## C 0.5987 0.20686 0.0010343      0.0030714
## S 0.8300 0.05039 0.0002519      0.0005409
## p 0.7849 0.19287 0.0009643      0.0033978
##
## 2. Quantiles for each variable:
##
##     2.5%    25%    50%    75%  97.5%
## C 0.2238 0.4337 0.6118 0.7661 0.9442
## S 0.7343 0.7951 0.8292 0.8657 0.9267
## p 0.2054 0.7250 0.8395 0.9160 0.9902
```

Our point estimates here are a bit high for specificity and prevalence, but the ranges are very close in all cases.

Let's put together the data that we have so far to replicate figure 2 from JGC.

The first step will be putting `coda` data into data frames. We have to do this because there's no package I'm aware of that allows plotting different parameters from different models in the same plot. Plus, plotting in base `R` gives us a bit of the 1995 feeling that JGC has.

```r
stool_samples_df <- as.data.frame(as.matrix(stool_samples))
sero_samples_df  <- as.data.frame(as.matrix(sero_samples))
plot(density(stool_samples_df$S),
     xlim = c(0, 1),
     ylim = c(0, 14),
     xlab = "Sensitivities and Specificities",
     ylab = "Posterior density",
     lty  = 1,
     main = "")
lines(density(sero_samples_df$S),
      lty = 2)
lines(density(stool_samples_df$C),
      lty = 3)
lines(density(sero_samples_df$C),
      lty = 4)
legend(0, 14,
       legend = c("Sensitivity of stool examination",
                  "Specificity of stool examination",
                  "Sensitivity of serologic test",
                  "Specificity of serologic test"),
       lty=1:4,
       cex=0.8)
```



Looking good!

Now to explore the full dataset in which both tests were administered to all patients. We'll be able to compare the test characteristics when both tests are administered versus just one, as well as to update our estimate of the prevalence of *Strongyloides*. This approach uses the Hui-Walter paradigm, and the code again borrows heavily from the Lewis / Gorgerson paper linked above. However, you can find many papers that use this approach (and provide code) if you search for "Hui-Walter paradigm" in Google Scholar or similar.

Our main objective in running this is to estimate the prevalence based on the results of both tests. In examining the joint probability of the two tests, though, the likelihood data isn't drawn from a distrubtion but rather from the observations themselves. The `L` vector below, then, determines what combination of test results a given observation should contribute its likelihood to. And the vector of ones uses a trick from the WinBUGS days to make sure that these likelihood values contribute to the model.

The matrices we'll construct to contain the different joint probabilities are constructed within the model itself by the `var` command.

For this section, we'll consider the stool examination to be test 1 and the serologic test to be test 2. Therefore, the parameter `C1`, for example, will be the specificity of the stool exam.

```
ones  <- rep(1, times = N)
tests <- data.frame("stool"    = stool,
                    "serology" = serology)

model_string_two_test <- "
var pr[N], q[N,4]
model {
  for(i in 1:N){
    q[i, 1] <- p*(S1*S2) + (1-p)*((1-C1)*(1-C2)) #Pr(+,+)
    q[i, 2] <- p*(S1*(1-S2)) + (1-p)*((1-C1)*C2) #Pr(+,-)
    q[i, 3] <- p*((1-S1)*S2) + (1-p)*(C1*(1-C2)) #Pr(-,+)
    q[i, 4] <- p*((1-S1)*(1-S2)) + (1-p)*(C1*C2) #Pr(-,-)

  L[i] <- equals(tests[i,1],1) * equals(tests[i,2],1) * q[i,1]
        + equals(tests[i,1],1) * equals(tests[i,2],0) * q[i,2]
        + equals(tests[i,1],0) * equals(tests[i,2],1) * q[i,3]
        + equals(tests[i,1],0) * equals(tests[i,2],0) * q[i,4]

  pr[i] <- L[i] / 1
  ones[i] ~ dbern(pr[i])
  }

  S1 ~ dbeta(4.44, 13.31)
  C1 ~ dbeta(71.25, 3.75)
  S2 ~ dbeta(21.96, 5.49)
  C2 ~ dbeta(4.1, 1.76)
  p ~ dunif(0,1)
}
"

writeLines(model_string_two_test, con = "two_test_model.txt")
jags_two_test <- jags.model("two_test_model.txt",
                        data     = list('tests' = tests,
                                        'ones'  = ones,
                                        'N'     = N),
                        n.chains = 4,
                        n.adapt  = 1000)
```

```
## Compiling model graph
##    Declaring variables
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 162
##    Unobserved stochastic nodes: 5
##    Total graph size: 5577
##
## Initializing model
```

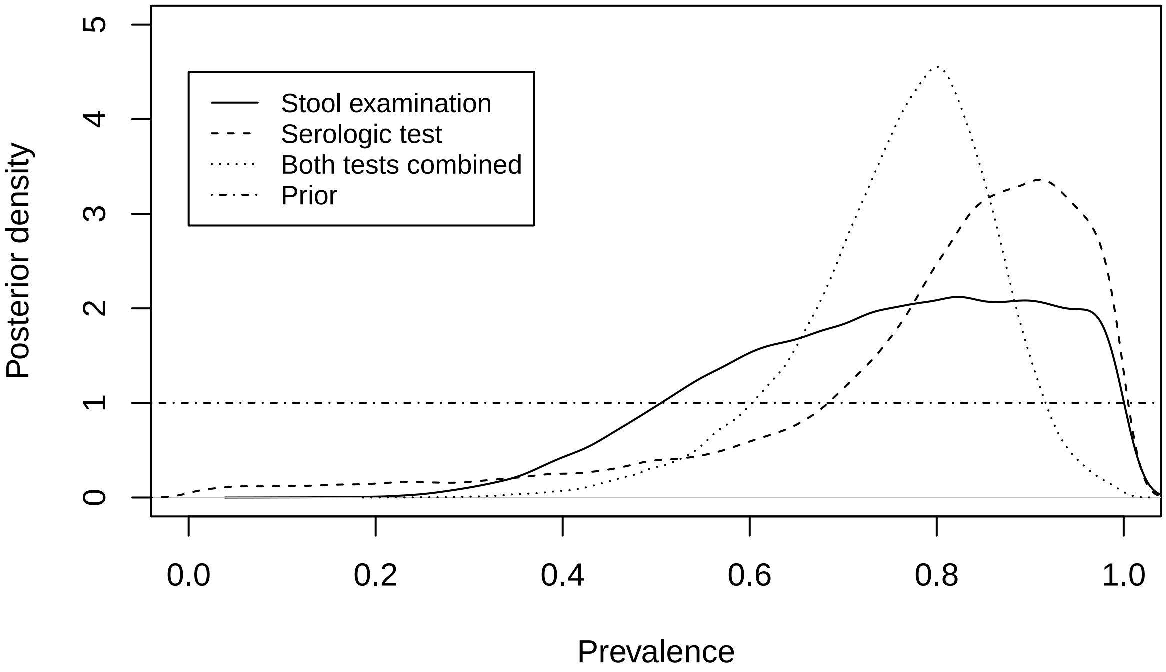
```
update(jags_two_test, 10000) #burn in

two_test_samples <- coda.samples(jags_two_test,
                             c("p", "S1", "C1", "S2", "C2"),
                             n.iter = 10000)
summary(two_test_samples)
```

```
##
## Iterations = 11001:21000
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##      Mean      SD  Naive SE Time-series SE
## C1 0.9571 0.02136 0.0001068      0.0001624
## C2 0.6861 0.16161 0.0008081      0.0022119
## S1 0.3108 0.05348 0.0002674      0.0006835
## S2 0.8835 0.04191 0.0002095      0.0004144
## p  0.7580 0.10324 0.0005162      0.0015885
##
## 2. Quantiles for each variable:
##
##      2.5%    25%    50%    75%  97.5%
## C1 0.9065 0.9453 0.9603 0.9727 0.9886
## C2 0.3680 0.5652 0.6950 0.8155 0.9570
## S1 0.2222 0.2743 0.3049 0.3404 0.4347
## S2 0.7911 0.8573 0.8872 0.9141 0.9538
## p  0.5114 0.7016 0.7727 0.8294 0.9238
```

Again, really close to the original JGC values for both the point estimates and the 95% credible intervals.

Finally, we have enough information to replicate Figure 1 from JGC. We'll need to extract the prevalence data from the two-test model and plot that on the same plot as the prevalence estimates from the previous one-test models.

```r
two_test_samples_df  <- as.data.frame(as.matrix(two_test_samples))
plot(density(stool_samples_df$p),
     xlim = c(0, 1),
     ylim = c(0, 5),
     xlab = "Prevalence",
     ylab = "Posterior density",
     lty  = 1,
     main = "")
lines(density(sero_samples_df$p),
      lty = 2)
lines(density(two_test_samples_df$p),
      lty = 3)
abline(h  = 1,
      lty = 4)
legend(0, 4.5,
       legend = c("Stool examination",
                  "Serologic test",
                  "Both tests combined",
                  "Prior"),
       lty=1:4,
       cex=0.8)
```



Again, this looks remarkably similar to the published JGC version.

I hope this was helpful and that it provides a template for your own JAGS work. Feel free to contact me with any questions and I'll help if I can.