Jags Error Messages

Heather Gaya, heather.e.gaya@gmail.com

10/5/2020

Contents

JAGS can produce all sorts of fun error messages! They can often seem confusing when first starting out, but with a little practice most of them *eventually* make sense. Key word most :)

If you have any others that you've come across and don't understand what they come from, let me know!

First, like all good examples, we need a model. Here's a simple occupancy model, that I am calling "modelstring.correct"

```
modelstring.correct = "
model {

# Prior for occupancy parameter
psi ~ dunif(0,1)

# Prior for detection probability
p ~ dunif(0,1)

for(i in 1:nSites) {
  # Latent presence/absence
  z[i] ~ dbern(psi)
  for(j in 1:nOccasions) {
    # Model for the data
    y[i,j] ~ dbern(z[i]*p)
  }
}
sitesOccupied <- sum(z)

}
"
```

And here's the correct way to give JAGS some data and other information it needs

```
y <- read.csv("cawa_data_2017_occu.csv", row.names = 1) #grab data
jags.data <- list(y=as.matrix(y), nSites=109, nOccasions=4)

jags.inits <- function() {
  list(psi=runif(1), p=runif(1), z=rep(1, 109))
}

jags.pars <- c("psi", "p", "sitesOccupied")

library(runjags)
jags.post.samples <- run.jags(model = modelstring.correct,
                              data=jags.data, inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, method = "parallel")
```

```
## Calling 3 simulations using the parallel method...
## All chains have finished
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Finished running the simulation
```

```
summary(jags.post.samples)
```

```
##                   Lower95     Median   Upper95       Mean         SD Mode
## psi             0.0620213  0.1149695  0.178067  0.1172822 0.03051320   NA
## p               0.6903840  0.8014635  0.901834  0.7975120 0.05651575   NA
## sitesOccupied 12.0000000 12.0000000 12.000000 12.0330000 0.18863615   12
##                       MCerr MC%ofSD SSeff        AC.10     psrf
## psi            0.0003939237     1.3  6000  0.012170561 1.000038
## p              0.0010174190     1.8  3086 -0.000844575 1.001021
## sitesOccupied 0.0024901440     1.3  5739  0.012895441 1.009619
```

There are three types of errors you can get when working with JAGS - problems with the model itself, problems with the data/initial values/monitors that you give to to run the model, and problems with R.

The above model is written correctly. Given the right data and information, it will run happily without error. Let's see what happens when we break it in various ways:

Model Writing Errors

Unknown Function

First, let's try to draw something from a distribution using the wrong symbol - a "<-" instead of a "~"

We'll change psi ~ dunif(0,1) to psi <- dunif(0,1).

```
jags.post.samples <- run.jags(model = modelstring.sad1,
                              data=jags.data, inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, silent.jags = F)
```

```
## Loading required namespace: rjags

## Compiling rjags model...

## Error: The following error occured when compiling and adapting the model using rjags:
##  Error in rjags::jags.model(model, data = dataenv, inits = inits, n.chains = length(runjags.object$e
##   RUNTIME ERROR:
## Compilation error on line 3.
## Unknown function: dunif
##
##
##
## It may help to use failed.jags(c('model','data','inits')) to see model/data/inits syntax with line n
```

Obviously we know that the uniform distribution is allowed in JAGS so this error can be a head-scratcher for sure. But since we used the assignment symbol instead of the distribution symbol, JAGS is trying to find the function dunif() - which doesn't exist. This can also happen if you try to use an R command instead of a JAGS command, so make sure you know what language you're writing in!

Syntax Error

What about if we accidentally put in an extra symbol somewhere? Let's draw y[i,j] ~ dbern(z[i]p) instead of y[i,j] ~ dbern(z[i]*p)

```
jags.post.samples <- run.jags(model = modelstring.sad3,
                              data=jags.data, inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, silent.jags = F)
```

```
## Compiling rjags model...

## Error: The following error occured when compiling and adapting the model using rjags:
##  Error in rjags::jags.model(model, data = dataenv, inits = inits, n.chains = length(runjags.object$e
##
## Error parsing model file:
## syntax error on line 11 near ")"
##
##
##
## It may help to use failed.jags(c('model','data','inits')) to see model/data/inits syntax with line n
```

Oh no, a syntax error! It can often help to ask your JAGS package to spit the model back out but with line numbers so that you know what "line 11" means. Remember that JAGS models don't count empty lines as true "lines" so if you're trying to count manually on a long model you can quickly lose track.

In the runjags package you can use the "failed.jags" command to look at the model numbers. There are similar commands in the other JAGS packages as well. Alternatively, you can open up your model in a .txt file and look at the lines that way.

```
failed.jags("model")
```

```
##
## JAGS model syntax:
##
```

```
## 1   |  model{
## 2   |  # Prior for occupancy parameter
## 3   |  psi ~ dunif(0,1)
## 4   |  # Prior for detection probability
## 5   |  p ~ dunif(0,1)
## 6   |  for(i in 1:nSites) {
## 7   |  # Latent presence/absence
## 8   |  z[i] ~ dbern(psi)
## 9   |  for(j in 1:nOccasions) {
## 10  |  # Model for the data
## 11  |  y[i,j] ~ dbern(z[i]*p*)
## 12  |  }
## 13  |  }
## 14  |  sitesOccupied <- sum(z)
## 15  |  }
```

Now we can clearly see that line 11 is the line with the extra asterisk in it.

Unmatched number of { and }

What if we misplace a bracket? Let's delete one from our model

```r
jags.post.samples <- run.jags(model = modelstring.sad2,
                              data=jags.data, inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, silent.jags = F)
```

```
## Error: Unmatched number of { and } in the specified model file - ensure all (uncommented) braces are
```

Luckily this error message is pretty obvious.

Attempt to Redefine Node

One thing that JAGS does not like is attempts to define a parameter in two ways at the same time.

Let's say we accidentally model p two ways in the same model. This can happen if you're testing out your model and holding things constant and forget to fix the changes later.

We model p as p~dunif(0,1) and p <- .5

```r
modelstring.sad4 = "
model {

# Prior for occupancy parameter
psi ~ dunif(0,1)

# Prior for detection probability

p ~ dunif(0,1)
p <- .5

for(i in 1:nSites) {
  # Latent presence/absence
  z[i] ~ dbern(psi)
  for(j in 1:nOccasions) {
    # Model for the data
    y[i,j] ~ dbern(z[i]*p)
  }
}

sitesOccupied <- sum(z)

}
"
```

```
jags.post.samples <- run.jags(model = modelstring.sad4,
                              data=jags.data, inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, silent.jags = F)
```

```
## Compiling rjags model...
```

```
## Error: The following error occured when compiling and adapting the model using rjags:
##  Error in rjags::jags.model(model, data = dataenv, inits = inits, n.chains = length(runjags.object$er
##   RUNTIME ERROR:
## Compilation error on line 5.
## Attempt to redefine node p[1]
##
##
##
## It may help to use failed.jags(c('model','data','inits')) to see model/data/inits syntax with line nu
```

Another way to get this same error is to put a constant prior inside a loop. For instance, in our model p is just one value. It doesn't matter what site you're at, p is p is p. But if we stick it inside a loop without indexing it, it tries to redraw p every time it goes through the loop.

```
modelstring.sad4.2 = "
model {

# Prior for occupancy parameter
psi ~ dunif(0,1)

# Prior for detection probability is now INSIDE the loop (bad)

for(i in 1:nSites) {
p ~ dunif(0,1)
  # Latent presence/absence
  z[i] ~ dbern(psi)
  for(j in 1:nOccasions) {
    # Model for the data
    y[i,j] ~ dbern(z[i]*p)
  }
}

sitesOccupied <- sum(z)

}
"
```

```
jags.post.samples <- run.jags(model = modelstring.sad4.2,
                              data=jags.data, inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, silent.jags = F)
```

```
## Compiling rjags model...

## Error: The following error occured when compiling and adapting the model using rjags:
##  Error in rjags::jags.model(model, data = dataenv, inits = inits, n.chains = length(runjags.object$en
##   RUNTIME ERROR:
## Compilation error on line 6.
## Attempt to redefine node p[1]
##
##
##
## It may help to use failed.jags(c('model','data','inits')) to see model/data/inits syntax with line nu
```

Cannot Insert Node

Sometimes when switching back and forth between models, you might get the dimensions of an object wrong. Maybe in a previous model you modeled detection probability as a vector of probabilities, one for each time period in your survey.

So you accidentally have p ~ dunif(0,1) in one place and p[3] <- .34 in another place.

```
modelstring.sad5 = "
model {

# Prior for occupancy parameter
psi ~ dunif(0,1)

# Prior for detection probability
p ~ dunif(0,1)
p[3] <- .34

for(i in 1:nSites) {
  # Latent presence/absence
  z[i] ~ dbern(psi)
  for(j in 1:nOccasions) {
    # Model for the data
    y[i,j] ~ dbern(z[i]*p)
  }
}

sitesOccupied <- sum(z)

}
"
```

This particular way of writing the model will give you an error saying that there is a dimension mismatch and you cannot insert the value into the node.

```
jags.post.samples <- run.jags(model = modelstring.sad5,
                                data=jags.data, inits=jags.inits,
                                  monitor=jags.pars,
                                  n.chains=3, adapt=100, burnin=100,
                                  sample=2000, silent.jags = F)
```

```
## Compiling rjags model...

## Error: The following error occured when compiling and adapting the model using rjags:
##  Error in rjags::jags.model(model, data = dataenv, inits = inits, n.chains = length(runjags.object$en
##   RUNTIME ERROR:
## Cannot insert node into p[1:3]. Dimension mismatch
##
##
##
## It may help to use failed.jags(c('model','data','inits')) to see model/data/inits syntax with line nu
```

Failed Check for Discrete-Valued Parameters in Distribution

An error I recently encountered comes from mixing up the syntax of R and JAGS commands. In T, a random draw from a binomial distribution is done via rbinom(n,size,prob). But in JAGS, you use dbin(prob,size). If you do it backwards, JAGS yells.

Here's we'll incorrectly draw z[i] ~dbin(1,psi) instead of z[i] ~dbin(psi,1) (equivalent to dbern(psi))

```
modelstring.sad6 = "
model {

# Prior for occupancy parameter
psi ~ dunif(0,1)

# Prior for detection probability
p ~ dunif(0,1)

for(i in 1:nSites) {
  # Latent presence/absence
  z[i] ~dbin(1,psi)
  for(j in 1:nOccasions) {
    # Model for the data
    y[i,j] ~ dbern(z[i]*p)
  }
}

sitesOccupied <- sum(z)

}
"
```

```
jags.post.samples <- run.jags(model = modelstring.sad6,
                              data=jags.data, inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, silent.jags = F)
```

```
## Compiling rjags model...

## Error: The following error occured when compiling and adapting the model using rjags:
##  Error in rjags::jags.model(model, data = dataenv, inits = inits, n.chains = length(runjags.object$en
##   RUNTIME ERROR:
## Failed check for discrete-valued parameters in distribution dbin
##
##
##
```

```
## It may help to use failed.jags(c('model','data','inits')) to see model/data/inits syntax with line nu
```

Data Errors

So let's say we get the correct model written out, but then we don't give JAGS the correct information to let it run. What happens? New errors! Here's the correct model for reference.

```
modelstring.correct = "
model {

# Prior for occupancy parameter
psi ~ dunif(0,1)

# Prior for detection probability
p ~ dunif(0,1)

for(i in 1:nSites) {
  # Latent presence/absence
  z[i] ~ dbern(psi)
  for(j in 1:nOccasions) {
    # Model for the data
    y[i,j] ~ dbern(z[i]*p)
  }
}

sitesOccupied <- sum(z)

}
"
```

Let's start with my "favorite" error, the bane of many an R-users existence, the "parent node" problem.

Node Inconsistent with Parents

Let's say we run this model and we don't provide any initial values. Sometimes you can get away with this, because JAGS will try to choose reasonable starting values and it is *possible* that these values will work great. But more often that not, the starting values do not make any sense and JAGS is confused.

```
library(runjags)
jags.post.samples <- run.jags(model = modelstring.correct,
                              data=jags.data,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, method = "parallel", silent.jags = F)
```

```
## Calling 3 simulations using the parallel method...
## Following the progress of chain 1 (the program will wait for all chains
## to finish before continuing):
## Welcome to JAGS 4.3.0 on Mon Oct  5 18:18:57 2020
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 436
##    Unobserved stochastic nodes: 111
##    Total graph size: 662
## . Reading parameter file inits1.txt
## . Initializing model
## Error in node y[1,1]
## Node inconsistent with parents
## Deleting model
## . Adaptation skipped: model is not in adaptive mode.
## . Updating 100
## -------------------------------------------------| 100
## Can't update. No model!
## Deleting model
##
## . Can't set monitor. No model!
## . Can't set monitor. No model!
## . Can't set monitor. No model!
## . Updating 2000
## -------------------------------------------------| 2000
## Can't update. No model!
## Deleting model
##
## . No model
## . Can't dump CODA output. No model!
## . Can't dump samplers. No model!
## . Updating 0
## Can't update. No model!
## Deleting model
## . Deleting model
## .
## All chains have finished
## Note: the model did not require adaptation
```

```
## Error in runjags.readin(directory = startinfo$directory, silent.jags = silent.jags, : All the simula
```

```
## Note: Either one or more simulation(s) failed, or there was an error in
## processing the results.  You may be able to retrieve any successful
## simulations using:
## results.jags("/private/var/folders/v4/v904tsfx66zc2rwrthj56_y80000gn/T/RtmpnM5ckL/runjagsfiles153f37
## recover.chains=TRUE)
## See the help file for that function for possible options.
## To remove failed simulation folders use cleanup.jags() - this will be
## run automatically when the runjags package is unloaded
```

The first issue with data errors, at least in this particular R package, is that the errors are very long. This is because the model is fine and JAGS starts by compiling the model, so it runs through all that before it realizes something is wrong.

In the above, the exact error is "Error in node y[1,1] Node inconsistent with parents"

In normal human english, this means something like "the value for y[1,1] doesn't make any sense given the other data. Help?"

First step, we ask "well, okay, what *IS* y[1,1]?"

```r
y[1,1]
```

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

Okay, great. The value is 1. In our model, we draw y[i,j] ~ dbern(z[i]*p). We just said y[1,1] = 1, which means logically z[1] HAS to equal 1. But we didn't tell JAGS that z[1] = 1, so it randomly said "Well, looks like z[i] can be 0 or 1, so let's set it to 0." But since a bernoulli with probability 0 cannot result in a result of 1, it throws an error.

An easy fix for this issue is to set initial values that make sense. In this case, we can start by just setting all the z's equal to 1.

```r
jags.inits <- function() {
  list(z=rep(1, 109))
}
jags.post.samples <- run.jags(model = modelstring.correct,
                              data=jags.data, inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, silent.jags = F)
```

```
## Compiling rjags model...
## Calling the simulation using the rjags method...
## Adapting the model for 100 iterations...
## Burning in the model for 100 iterations...
## Running the model for 2000 iterations...
## Simulation complete
## Calculating summary statistics...
## Calculating the Gelman-Rubin statistic for 3 variables....
## Finished running the simulation
```

In this case, finding a logical initial value is really easy. In some cases this can take a very long time to resolve. For many classes of models, people have come up with brilliant ways to skip this frustration. But for others... it's just a pain.

This error can also happen if you provide an initial value that makes no sense. Like, p = 1.5 (in the model, we define p as a value between 0 and 1).

```r
jags.inits <- function() {
  list(z=rep(1, 109), p = 1.5)
}
jags.post.samples <- run.jags(model = modelstring.correct,
                              data=jags.data, inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, silent.jags = F)
```

```
## Compiling rjags model...

## Error: The following error occured when compiling and adapting the model using rjags:
##  Error in rjags::jags.model(model, data = dataenv, inits = inits, n.chains = length(runjags.object$er
##   Error in node y[1,1]
## Invalid parent values
##
##
##
## It may help to use failed.jags(c('model','data','inits')) to see model/data/inits syntax with line nu
```

Notice that the actual problem here is NOT y[1,1] - we know that if we removed our bad initial value for p that the model would run. We also know that p cannot be 1.5, it just isn't possible. I don't have a brilliant suggestion here, this particular error just takes time to unravel. The best solution is to follow the path from data to prior and make sure all the values are a) possible and b) reasonable.

Cannot Set Value of Non-Variable Node

Speaking of initial values, JAGS does not approve of setting values on things that do not come from distributions. So our variable "sitesOccupied", which is really just adding up all the sites where z = 1, cannot have its own initial values. Instead, we have to put the initial values on z.

```
jags.inits <- function() {
  list(z=rep(1, 109), sitesOccupied = 10)
}
jags.post.samples <- run.jags(model = modelstring.correct,
                              data=jags.data, inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, silent.jags = F)
```

```
## Compiling rjags model...

## Error: The following error occured when compiling and adapting the model using rjags:
##  Error in setParameters(init.values[[i]], i) : Error in node sitesOccupied
## Cannot set value of non-variable node
##
##
##
## It may help to use failed.jags(c('model','data','inits')) to see model/data/inits syntax with line nu
```

Unknown Variable

Sometimes, by which I mean very often, you might forget to give jags some data.

```
jags.inits <- function() {
  list(z=rep(1, 109))
}
jags.data <- list(y=as.matrix(y), nOccasions=4)
jags.post.samples <- run.jags(model = modelstring.correct,  data=jags.data,
                              inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, silent.jags = F)
```

```
## Compiling rjags model...

## Error: The following error occured when compiling and adapting the model using rjags:
##  Error in rjags::jags.model(model, data = dataenv, inits = inits, n.chains = length(runjags.object$en
##   RUNTIME ERROR:
## Compilation error on line 6.
## Unknown variable nSites
## Either supply values for this variable with the data
## or define it  on the left hand side of a relation.
##
##
##
## It may help to use failed.jags(c('model','data','inits')) to see model/data/inits syntax with line nu
```

In this case, when JAGS gets down to the part of the model that loops over all the sites, it doesn't know what "nSites" means unless we give it that data in the data argument OR we hardcode it into the model somewhere.

Cannot Resolve Node and Cannot Evaluate Subset Expression for sitesOccupied

For these errors, we have to make our model a little more complex. Let's model some site covariates. Now psi is no longer just one value - instead it might change based on some site characteristic, like rainfall.

```
modelstring.correct2 = "
model {

# Priors for occupancy parameters
psi.beta0 ~ dunif(-3,3)
psi.beta1 ~ dunif(-3,3)

# Prior for detection probability
p ~ dunif(0,1)

for(i in 1:nSites) {
logit(psi[i]) <- psi.beta0+psi.beta1*rainfall[i]
  # Latent presence/absence
  z[i] ~ dbern(psi)
  for(j in 1:nOccasions) {
    # Model for the data
    y[i,j] ~ dbern(z[i]*p)
  }
}

sitesOccupied <- sum(z)

}
"
```

Importantly, the above model does not have any distribution for rainfall. It assumes we have data for all of our sites. But what if we have an NA in there?

```
rainfall = runif(109, 0, 10) #just making up some rainfall data
rainfall[c(5,15,20)] <- NA #3 of them are now NA
jags.data <- list(y=as.matrix(y), nSites=109, nOccasions=4, rainfall = rainfall)
jags.inits <- function() {
  list(z=rep(1, 109))
}
```

```
jags.post.samples <- run.jags(model = modelstring.correct2,
                              data=jags.data, inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, method = "parallel", silent.jags = F)
```

```
## Calling 3 simulations using the parallel method...
## Following the progress of chain 1 (the program will wait for all chains
## to finish before continuing):
## Welcome to JAGS 4.3.0 on Mon Oct  5 18:19:00 2020
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## RUNTIME ERROR:
## Compilation error on line 16.
## Cannot evaluate subset expression for sitesOccupied
## Deleting model
## . Reading parameter file inits1.txt
## Can't set RNG name. No model!
## Can't set initial values. No model!
## . Can't initialize. No model!
## . Adaptation skipped: model is not in adaptive mode.
## . Updating 100
## -------------------------------------------------| 100
## Can't update. No model!
## Deleting model
##
## . Can't set monitor. No model!
## . Can't set monitor. No model!
## . Can't set monitor. No model!
## . Updating 2000
## -------------------------------------------------| 2000
## Can't update. No model!
## Deleting model
##
## . No model
## . Can't dump CODA output. No model!
## . Can't dump samplers. No model!
## . Updating 0
## Can't update. No model!
## Deleting model
## . Deleting model
## .
## All chains have finished
## Note: the model did not require adaptation
```

```
## Error in runjags.readin(directory = startinfo$directory, silent.jags = silent.jags, : All the simula
```

```
## Note: Either one or more simulation(s) failed, or there was an error in
## processing the results.  You may be able to retrieve any successful
## simulations using:
## results.jags("/private/var/folders/v4/v904tsfx66zc2rwrthj56_y80000gn/T/RtmpnM5ckL/runjagsfiles153f34
## recover.chains=TRUE)
## See the help file for that function for possible options.
## To remove failed simulation folders use cleanup.jags() - this will be
## run automatically when the runjags package is unloaded
```

So the error we get is “Cannot evaluate subset expression for sitesOccupied.” That doesn’t really make a lot of sense because we said sitesOccupied <- sum(z), which seems pretty straightforward. In this case, a *SECOND* error is actually lurking. If you ever get an error that you can’t explain, try commenting out the part of the model with the error and see if it runs. It can really help narrow down the problem.

Let’s comment out sitesOccupied from the model:

```
modelstring.correct3 = "
model {

# Priors for occupancy parameters
psi.beta0 ~ dunif(-3,3)
psi.beta1 ~ dunif(-3,3)

# Prior for detection probability
p ~ dunif(0,1)

for(i in 1:nSites) {
logit(psi[i]) <- psi.beta0+psi.beta1*rainfall[i]
  # Latent presence/absence
  z[i] ~ dbern(psi)
  for(j in 1:nOccasions) {
    # Model for the data
    y[i,j] ~ dbern(z[i]*p)
  }
}

#sitesOccupied <- sum(z)

}
"
```

```
jags.post.samples <- run.jags(model = modelstring.correct3,
                              data=jags.data, inits=jags.inits,
                                monitor=jags.pars,
                                n.chains=3, adapt=100, burnin=100,
                                sample=2000, method = "parallel", silent.jags = F)
```

```
## Calling 3 simulations using the parallel method...
## Following the progress of chain 1 (the program will wait for all chains
## to finish before continuing):
## Welcome to JAGS 4.3.0 on Mon Oct  5 18:19:00 2020
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## RUNTIME ERROR:
## Unable to resolve the following parameters:
## rainfall[5] (line 8)
## rainfall[15] (line 8)
## rainfall[20] (line 8)
## Either supply values for these nodes with the data
## or define them on the left hand side of a relation.
## Deleting model
## . Reading parameter file inits1.txt
## Can't set RNG name. No model!
## Can't set initial values. No model!
## . Can't initialize. No model!
## . Adaptation skipped: model is not in adaptive mode.
## . Updating 100
## -------------------------------------------------| 100
## Can't update. No model!
## Deleting model
##
## . Can't set monitor. No model!
## . Can't set monitor. No model!
## . Can't set monitor. No model!
## . Updating 2000
## -------------------------------------------------| 2000
## Can't update. No model!
## Deleting model
##
## . No model
## . Can't dump CODA output. No model!
## . Can't dump samplers. No model!
## . Updating 0
## Can't update. No model!
## Deleting model
## . Deleting model
## .
## All chains have finished
## Note: the model did not require adaptation
```

```
## Error in runjags.readin(directory = startinfo$directory, silent.jags = silent.jags, : All the simula
```

```
## Note: Either one or more simulation(s) failed, or there was an error in
## processing the results.  You may be able to retrieve any successful
## simulations using:
## results.jags("/private/var/folders/v4/v904tsfx66zc2rwrthj56_y80000gn/T/RtmpnM5ckL/runjagsfiles153f31
## recover.chains=TRUE)
## See the help file for that function for possible options.
## To remove failed simulation folders use cleanup.jags() - this will be
## run automatically when the runjags package is unloaded
```

Ahh, there it is! The 'real' error in this case was "Unable to resolve the following parameters" which comes from rainfall having NA's and no information for JAGS about reasonable values to fill in for the NA's.

Note that if you actually have NA's you can either remove them from your data, or model them from a distribution bound by reasonable ranges of the covariate. This assumes that the distribution of the covariate in your data is the same as the distribution of the covariate at your study site or in your population, which is not always a fair assumption. So be careful out there… NA's are tricky buggers!

Other Errors

Do you have any other strange JAGS errors you've come across? Let me know! Hopefully this manual will evolve with time as new and exciting errors make their way across my R console :)