Known Fate Models With Model Selection

Heather Gaya

11/23/2020

I've recently been helping a friend work on a known-fate model for her dissertation and I figured that other people could benefit from what we've learned during this process! I'd never actually run a Bayesian known-fate model until a few weeks ago so I was pleasantly surprised to learn they're pretty easy but surprisingly powerful!

In this tutorial I'll show you how to run a known-fate model to estimate feral pig survival in JAGS and NIMBLE, using WAIC to choose the best covariate combination. The dataset itself is fake, but based off of some real data. I'll also show you how to graph the resulting survival curves!

As always, I'm assuming that everyone has downloaded JAGS and NIMBLE and has them setup on their computers. Before you use NIMBLE make sure R, and Rtools or Xcode are updated on your computer (otherwise a weird "shared library" error can come up). JAGS is downloadable here: https://sourceforge.net/projects/mcmc-jags/ and NIMBLE can be found here: https://r-nimble.org/download

Please send any questions or suggestions to heather.e.gaya(at)gmail.com or find me on twitter: doofgradstudent

Contents

A Scenario

Known-fate models are traditionally done with game species, but they don't have to be. Common textbook examples of these models include tracking turkeys after they hatch, releasing ducks each year and seeing how long they live before hunters shoot them (and return the tags), or collaring baby fawns. The study system itself doesn't really matter, the key to these models is that you're interested in survival. In some ways, these models are very similar to CJS models but without worrying about detection probability.

The scenario I've been working with recently is feral pigs. Pregnant female pigs were located on site and tracked until they gave birth. The piglets were then tracked for 42 days after they were born, though obviously tracking stopped if they died. Various covariates were also measured, such as which sow they came from, the experience level of the sow (older pigs may produce more successful piglets), the color of the piglet, the sex of the piglet, and the mass of the mom, among others.

The data started in a messy format, so we'll go through the cleanup process as well just for funsies.

Here's the "messy" form of our (mostly fake) data:

	ID	Sow	Sow_mass	sex	color	year	mass	In.Date	Out.Date	censor
15	1	21	1110.582	M	notspotted	2017	348.471	2017-01-20	2017-01-23	0
44	2	21	1110.582	F	spotted	2017	242.959	2017-01-20	2017-03-03	1
75	3	21	1110.582	F	notspotted	2017	300.432	2017-01-20	2017-03-03	1
99	4	21	1110.582	M	notspotted	2017	309.505	2017-01-20	2017-03-03	1
6	5	13	1103.094	M	spotted	2017	323.359	2017-02-27	2017-03-13	0
9	6	13	1103.094	F	spotted	2017	314.161	2017-02-27	2017-03-06	0

In this dataset, we can see that the "Out.Date" is sometimes the time the animal died and sometimes when the study period ended. We can tell which is the case based on the "censor" column - 1 means the animal lived throughout the study period, 0 means it died before the 42 day study period was over.

Okay, so what do we do with this data? First let's talk about the model.

The Known-Fate Model

The true "meat" of a known fate model is really just two equations.

First, we assume some combination of individual and environmental factors affect the daily/monthly/time-step-of-interest survival probability. For now, let's call this daily survival. Like pretty much every other wildlife bayesian model, this should immediately make you think "ahh, we will be using a logit link and a linear model!"

Okay, so some stuff goes together to affect survival. And then every day/month/time-step, the world flips a coin and says "yep, today you lived" or "nope sorry you have died". Of course, you can't have "zombie pigs" so if you're dead, you have to stay dead. That tells us that there has to be something about the previous state of live/deadness in the model in order for us to know if the animal will be alive next time.

In math terms,

$$\phi_{it} = \frac{exp(\alpha_0 + X1_{it} + X2_{it} + \dots)}{exp(\alpha_0 + X1_{it} + X2_{it} + \dots) + 1}$$
$$\mu_{it} = \phi_{it} * y_{i(t-1)}$$
$$y_{it} \sim \text{Bern}(\mu_{it})$$

where ϕ_{it} is the probability of animal i surviving from time t to time $t+1$, α_0, $X1_{it}$ and $X2_{it}$ represent various covariates on survival and $y_{i(t-1)}$ is the alive/dead state of animal i in time $t-1$.

Believe it or not, that's literally the entire model!

So let's think about how to wrangle our data into a useful format.

Data Cleanup

For this type of model, we don't need too many pieces of information. If you haven't coded this model before, it's hard to know what you'll need so I'll just tell you for now and you'll see how it all comes into play below.

We will need: - the number of individuals in our study - individual covariates - numbered time periods with (ideally) equally spaced time intervals between them - a matrix of observations of each individuals alive/dead state at all time periods - the first time period an animal was seen at - the last time period an animal was seen at

Let's take a quick peek at our data again (available in a csv file in the github repository should you want to "run this with me"):

```
neos <- read.csv("pig_survival.csv", stringsAsFactors = F)
head(neos, n = 2)
##    X ID Sow Sow_mass sex      color year     mass    In.Date   Out.Date censor
## 1 15  1  21 1110.582   M notspotted 2017 348.4714 2017-01-20 2017-01-23      0
## 2 44  2  21 1110.582   F    spotted 2017 242.9589 2017-01-20 2017-03-03      1
```

I like to start with the dates and move on from there. First step, convert the dates to actual dates instead of characters.

```
neos$In.Date <- as.Date(neos$In.Date, format = "%Y-%m-%d")
neos$Out.Date <- as.Date(neos$Out.Date, format = "%Y-%m-%d")
```

Excellent. Now we can do math and make sequences of these dates much more easily. For this example, the interest is in daily survival and 42-day survival, so we want the occasions to be days. But since animals were born and died at random days, we don't have a convenient list of every day that exists between our start and end date of our data set. So we have to make one.

```
cap.dates <- sort(unique(c(neos$In.Date, neos$Out.Date)))
cap.dates <- seq(cap.dates[1], cap.dates[length(cap.dates)], by = 1)
occs <- length(cap.dates)
```

We end up with 1070 time periods, though obviously we don't have data for most of those.

Now we can grab what time period each individual was born (first seen) and the last time period we observed them in. We also need to get our "capture history" for the individuals, which is just a matrix of their alive/dead state at any given time. If we know they are alive (first time period to the second-to-last time period), we know their live state = 1. If they are dead, their live state = 0. If we don't know, it is an NA.

```
first <- last <- array(NA, dim = nrow(neos)) #first and last day we saw each bubs
surv.caps <-  matrix(data = NA, nrow = nrow(neos), ncol = occs) #cap.history
for(i in 1:nrow(neos)){ #for each individual
  first[i] <- which(cap.dates == neos$In.Date[i]) #first seen
  last[i] <- which(cap.dates == neos$Out.Date[i]) #last seen
  surv.caps[i,first[i]:last[i]] <- 1 #it was alive for its lifetime obviously
  if(neos$censor[i] == 0)# unless it died
  {surv.caps[i,last[i]] <- 0}
  #if it died, it should be dead at the last occasion that we saw it
}
```

We can check our work on the first individual.

```
neos[1, ]
##    X ID Sow Sow_mass sex      color year     mass    In.Date   Out.Date censor
## 1 15  1  21 1110.582   M notspotted 2017 348.4714 2017-01-20 2017-01-23      0
surv.caps[1, 1:10]
##  [1]  1  1  1  0 NA NA NA NA NA NA
```

Lastly, we can standardize our continuous covariates and make sure all our categorial variables are numeric.

```
neos$Sow_mass_s <- (neos$Sow_mass - mean(neos$Sow_mass))/sd(neos$Sow_mass)
neos$mass_s <- (neos$mass - mean(neos$mass))/sd(neos$mass)
# could also use the scale function
neos$color <- as.numeric(as.factor(neos$color))
neos$sex <- as.numeric(as.factor(neos$sex))
neos$Sow_numeric <- as.numeric(as.factor(neos$Sow))
```

Awesome, that's the hardest parts of our data wrangling done! Now on to coding.

Known-fate in JAGS

A lot of this model's difficulty is really just getting your data setup, which we saw in the last section.

First step, let's convert that math above into code. Remember that the ugly fraction is just the logit transform (plogis in R, logit(something) in JAGS).

Intercept Only

We'll start with a really simple linear model, one with no covariates. We can make it more complex later.

```
logit(phi) <- beta0
```

Gorgeous. Next let's add in μ and y which we know are going to change based on if the animal is alive or dead, so they have to be indexed by individual and by time.

```
logit(phi) <- beta0
for (i in 1:n.ind) {
    for (t in 1:end.time) {
        mu[i, t] <- phi * y[i, t - 1]
        y[i, t] ~ dbern(mu[i, t])
    }
}
```

Hmm, so we come across a problem. What time steps do we need to loop over? What happens if we are at a time step before our indiviual was born? And why do we need to calculate if the animal will survive if it's already dead?

The answer is, we don't. We can make a loop that *changes for each individual*. Assume that we can grab out which time step we first saw the animal in and then the last time we saw the animal (either because we stopped the study or it died). Then we can index our loop over only this time period for each individual! Of course, in the first time step, the animal was born so we don't really care about the first time step, just the second all the way through to the last time step.

Check this out:

```
logit(phi) <- beta0
for (i in 1:n.ind) {
    for (t in (first[i] + 1):last[i]) {
        mu[i, t] <- phi * y[i, t - 1]
        y[i, t] ~ dbern(mu[i, t])
```

```
    }
}
```

This variable indexing is super cool and save a bunch of computation time. All that's left now is to give $beta0$ a prior and we can start thinking about running the model.

```
modelstring.neos_null = "
model {
logit(phi) <- beta0
for (i in 1:n.ind){
  for(t in (first[i]+1):last[i]){
    mu[i,t] <- phi*y[i,t-1]
    y[i,t] ~dbern(mu[i,t])
  }
}
beta0 ~ dunif(-6,6)
}
"
```

We are also going to add in one more line to this code so that we can calculate WAIC and do model selection. WAIC works by taking the log-likelihood of whatever distribution our data is drawn from. So since we have a bernoulli in this case, we'll need a log-bernoulli.

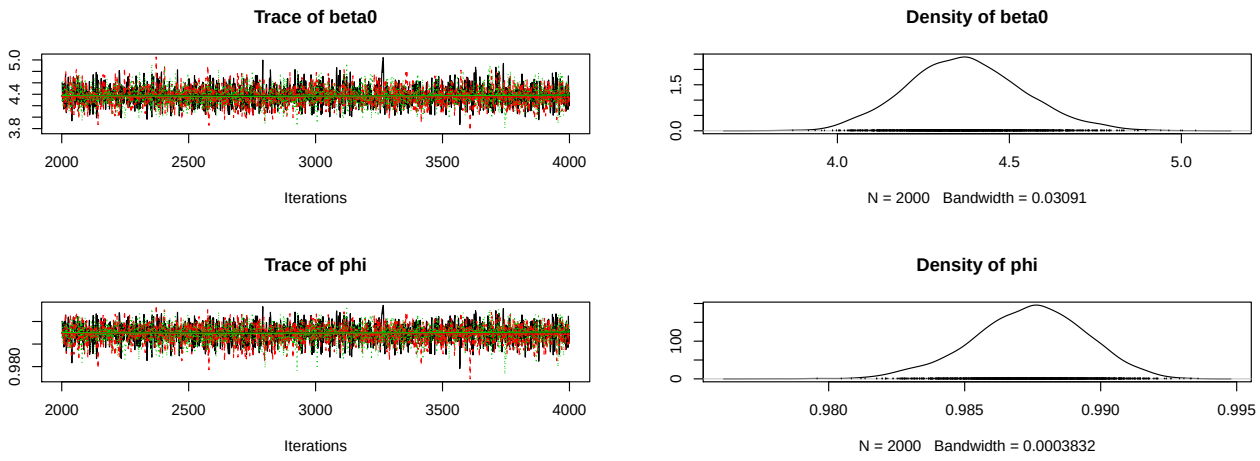
```
modelstring.neos_null = "
model {
logit(phi) <- beta0
for (i in 1:n.ind){
  for(t in (first[i]+1):last[i]){
    mu[i,t] <- phi*y[i,t-1]
    y[i,t] ~dbern(mu[i,t])
    loglike[i,t] <- logdensity.bern(y[i,t], mu[i,t]) #for WAIC
  }
}
beta0 ~ dunif(-6,6)
}
"
```

JAGS needs our data, our parameters of interest, some initial values and the name of our model.

```
jd <- list(n.ind = nrow(neos), y = surv.caps, first = first, last = last)
ji <- function() {
    list(beta0 = 0.5)
}
jp <- c("beta0", "phi", "loglike")  #make sure to monitor 'loglike' for WAIC
library(runjags)
piggies.null <- run.jags(model = modelstring.neos_null, monitor = jp, data = jd,
    inits = ji, n.chains = 3, burnin = 1000, sample = 2000, adapt = 1000, method = "parallel",
    thin = 1)
## Warning: You attempted to start parallel chains without setting different PRNG
## for each chain, which is not recommended. Different .RNG.name values have been
## added to each set of initial values.
## 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 Nov 23 12:59:06 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: 2919
##    Unobserved stochastic nodes: 1
##    Total graph size: 3227
## . Reading parameter file inits1.txt
## . Initializing model
## . Adapting 1000
## -------------------------------------------------| 1000
## +++++++++++++++++++++++++++++++++++++++++++++++++ 100%
## Adaptation successful
## . Updating 1000
## -------------------------------------------------| 1000
## ************************************************* 100%
## . . . . Updating 2000
## -------------------------------------------------| 2000
## ************************************************* 100%
## . . . . Updating 0
## . Deleting model
## All chains have finished
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Note: Summary statistics were not produced as there are >50 monitored
## variables
## [To override this behaviour see ?add.summary and ?runjags.options]
## FALSEFinished running the simulation
```

```
plot(piggies.null$mcmc[, c("beta0", "phi"), ])
```




```
summary(piggies.null$mcmc[, c("beta0", "phi"), ])
##
## Iterations = 2001:4000
## Thinning interval = 1
## Number of chains = 3
```

```
## Sample size per chain = 2000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean       SD  Naive SE Time-series SE
## beta0 4.3712 0.168838 2.180e-03      0.0028090
## phi   0.9874 0.002089 2.697e-05      0.0000334
##
## 2. Quantiles for each variable:
##
##          2.5%   25%    50%    75%  97.5%
## beta0 4.0524 4.256 4.3673 4.4787 4.7227
## phi   0.9829 0.986 0.9875 0.9888 0.9912
```

Awesome! Our model looks good. Our daily survival probability is estimated to be between 98% and 99%. You might notice that output summary is different than you might have seen before - that's okay! I didn't want to run summary on EVERY parameter because there's a lot of log-likelihood estimates and I don't care about them individually.

But obviously not ever piglet has the same survival probability. We know this model we just ran is super unrealistic. So let's add in some covariates, like a random effect for Sow and some effect of birth weight.

Random Effects

But now that we're adding in Sow and individual weight, we have to make sure that phi is indexed by i, since each individual has a different mom and a different birth weight. Also note that we have to use *nested indexing* for the categorical variable. We'll provide sow, mass, and n.sow as data.

```
modelstring.neos_Sow = "
model {

sigmaSow ~ dunif(0, .5)\t\t# Random effect SD between 0 and .5
  tauSow <- 1 / (sigmaSow * sigmaSow)
for(k in 1:n.sow){
  beta.Sow[k] ~ dnorm(0, tauSow)
}
for (i in 1:n.ind){
logit(phi[i]) <- beta0 + beta.Sow[sow[i]] + beta.mass*mass[i]
  for(t in (first[i]+1):last[i]){
    mu[i,t] <- phi[i]*y[i,t-1]
    y[i,t] ~dbern(mu[i,t])
    loglike[i,t] <- logdensity.bern(y[i,t], mu[i,t]) #for WAIC
  }
}
beta0 ~ dunif(-6,6)
beta.mass ~ dunif(-3,3)
}
"
```

Let's think about this for a minute. Let's say we tell the model, "hey individual 3 has sow #5". The model then says "okay, so the phi equation needs to use beta.Sow[5]". It goes up to the loop of beta.Sow's (1 beta for each Sow in our data) and says "okay, the effect on survival of this sow is a normal distribution, with mean 0 and standard deviation... something". With enough data, it will estimate this standard deviation for us. So why the tauSow? In JAGS, we can't just use sd in our normals, we have to use precision, which is 1/(sd^2).

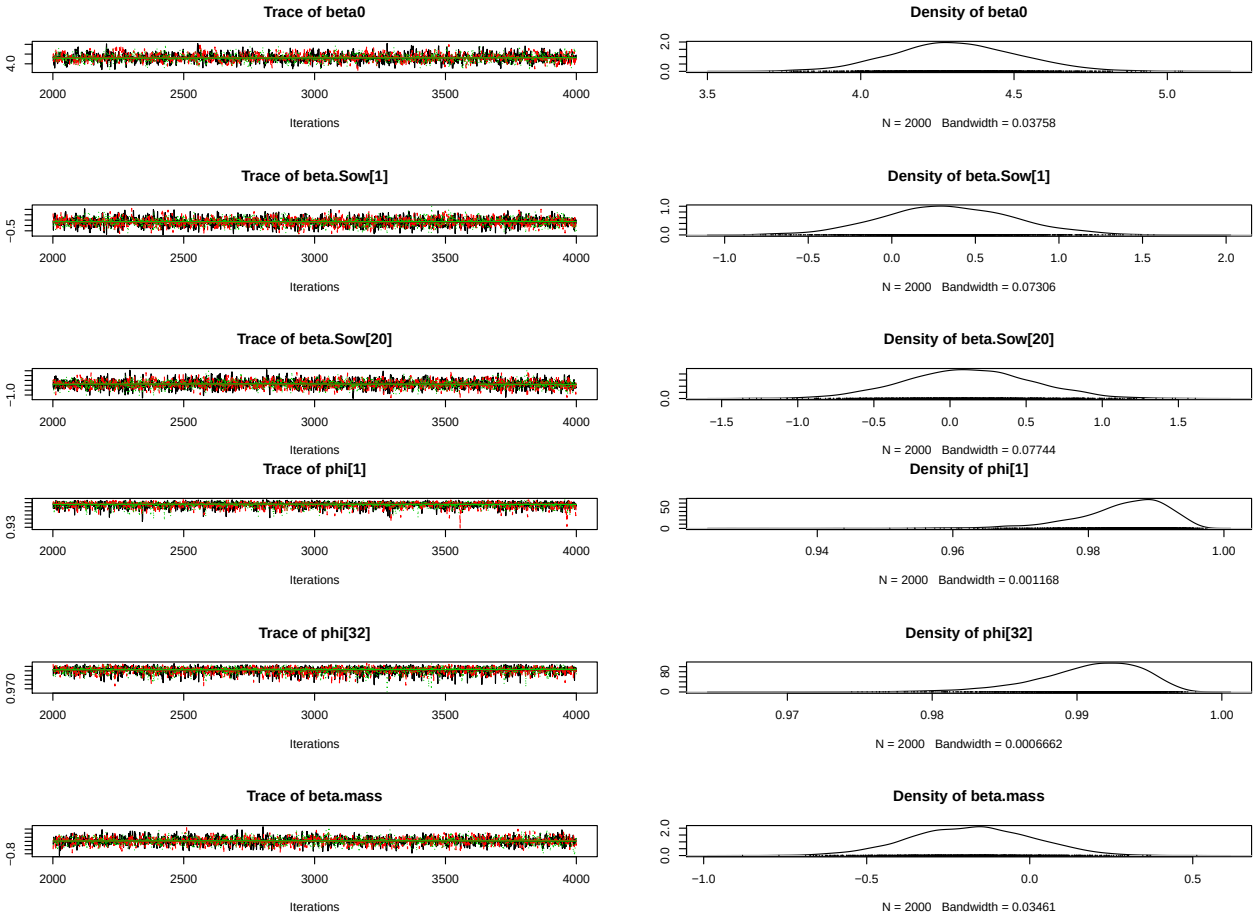
Let's see how this model looks when we run it.

```
n.sow = length(unique(neos$Sow))
jd <- list(n.ind = nrow(neos), y = surv.caps, first = first, last = last, sow = neos$Sow_numeric,
    n.sow = n.sow, mass = neos$mass_s)
ji <- function() {
    list(beta0 = 0.5, beta.Sow = runif(n.sow))
}
jp <- c("beta0", "phi", "loglike", "beta.Sow", "beta.mass")
# make sure to monitor 'loglike' for WAIC
library(runjags)
piggies.Sow <- run.jags(model = modelstring.neos_Sow, monitor = jp, data = jd, inits = ji,
    n.chains = 3, burnin = 1000, sample = 2000, adapt = 1000, method = "parallel",
    thin = 1)
## Warning: You attempted to start parallel chains without setting different PRNG
## for each chain, which is not recommended. Different .RNG.name values have been
## added to each set of initial values.
## 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 Nov 23 13:00:29 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: 2919
##    Unobserved stochastic nodes: 26
##    Total graph size: 3993
## . Reading parameter file inits1.txt
## . Initializing model
## . Adapting 1000
## -------------------------------------------------| 1000
## +++++++++++++++++++++++++++++++++++++++++++++++++ 100%
## Adaptation successful
## . Updating 1000
## -------------------------------------------------| 1000
## ************************************************* 100%
## . . . . . Updating 2000
## -------------------------------------------------| 2000
## ************************************************* 100%
## . . . Updating 0
## . Deleting model
## All chains have finished
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Note: Summary statistics were not produced as there are >50 monitored
## variables
## [To override this behaviour see ?add.summary and ?runjags.options]
## FALSEFinished running the simulation
```

Check some results. Rather than looking at all 100 survival probabilities and all 23 sow betas, I'll just graph

and summarize a few.

```
plot(piggies.Sow$mcmc[, c("beta0", "beta.Sow[1]", "beta.Sow[20]", "phi[1]", "phi[32]",
    "beta.mass"), ])
```




```
summary(piggies.Sow$mcmc[, c("beta0", "beta.Sow[1]", "beta.Sow[20]", "phi[1]", "phi[32]",
    "beta.mass"), ])
##
## Iterations = 2001:4000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 2000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                 Mean       SD  Naive SE Time-series SE
## beta0         4.3102 0.205603 2.654e-03      4.897e-03
## beta.Sow[1]   0.3341 0.392645 5.069e-03      6.911e-03
## beta.Sow[20]  0.1183 0.420191 5.425e-03      6.896e-03
## phi[1]        0.9849 0.007093 9.157e-05      1.160e-04
## phi[32]       0.9908 0.003771 4.869e-05      5.978e-05
## beta.mass    -0.1791 0.186003 2.401e-03      3.658e-03
##
## 2. Quantiles for each variable:
```

```
##
##                  2.5%      25%     50%      75%  97.5%
## beta0         3.9142  4.17439  4.3052  4.44500 4.7198
## beta.Sow[1]  -0.4274  0.07185  0.3271  0.60059 1.1270
## beta.Sow[20] -0.6962 -0.16318  0.1145  0.39448 0.9231
## phi[1]        0.9674  0.98144  0.9862  0.98985 0.9946
## phi[32]       0.9817  0.98876  0.9914  0.99355 0.9964
## beta.mass    -0.5451 -0.30557 -0.1778 -0.05068 0.1783
```

Individual 1 had the 20th Sow as his mom, and we can see that he has a slightly lower daily survival probability than individual 32, who had Sow 1 as his mom. This doesn't seem like that big a deal, but remember we've calculated *daily* survival. How likely are they to survive to day 42? Let's check:

```
0.9849^42
## [1] 0.5278005
0.9906^42
## [1] 0.6725578
```

Looks like individual 1 had about a 53% chance of making it to the end of the study period, but number 32 only had an 67% chance. If we look at our data, we can see that Sow 1 actually died during the study but 32 made it at least 42 days! As for mass, we can see that the beta's CI crosses 0. This means that according to this model, either the effect of mass is fairly small or it isn't actually that important for piglet daily survival.

Random Effect + Fixed Effect

Let's look at another model option. Maybe the sex is important or maybe there's also a year effect. Sex will be a categorical, but we can have the two sexes draw their betas independently, as a fixed effect. We might want the years to be a random effect to control the variation between years. Remember that JAGS uses precision not sd in normal distributions.

```
modelstring.neos_sexyear = "
model {

  beta.sex[1] ~ dnorm(0, 1.5) #female
  beta.sex[2] ~ dnorm(0, 1.5) #male

  sigmaSow ~ dunif(0, .5)\t\t# Random effect SD between 0 and .5
  tauSow <- 1 / (sigmaSow * sigmaSow)
for(k in 1:n.sow){
  beta.Sow[k] ~ dnorm(0, tauSow)
}

  sigmayear ~ dunif(0, .5)\t\t# Random effect SD
  tauyear <- 1 / (sigmayear * sigmayear)

for(t in 1:n.years){
  beta.year[t] ~ dnorm(0, tauyear)
}


for (i in 1:n.ind){
logit(phi[i]) <- beta0 + beta.sex[sex[i]] + beta.year[year[i]] + beta.Sow[sow[i]]
  for(t in (first[i]+1):last[i]){
    mu[i,t] <- phi[i]*y[i,t-1]
    y[i,t] ~dbern(mu[i,t])
```

```
    loglike[i,t] <- logdensity.bern(y[i,t], mu[i,t]) #for WAIC
  }
}
beta0 ~ dunif(-3,5)
}
"
```

We can also ask JAGS to estimate the 42 day survival probability for each sex for us so we don't have to calculate it later in R.

```
modelstring.neos_sexyear = "
model {

  beta.sex[1] ~ dnorm(0, 1.5) #female
  beta.sex[2] ~ dnorm(0, 1.5) #male

  sigmaSow ~ dunif(0, .5)\t\t# Random effect SD between 0 and .5
  tauSow <- 1 / (sigmaSow * sigmaSow)
for(k in 1:n.sow){
  beta.Sow[k] ~ dnorm(0, tauSow)
}

  sigmayear ~ dunif(0, .5)\t\t# Random effect SD
  tauyear <- 1 / (sigmayear * sigmayear)

for(t in 1:n.years){
  beta.year[t] ~ dnorm(0, tauyear)
}


for (i in 1:n.ind){
logit(phi[i]) <- beta0 + beta.sex[sex[i]] + beta.year[year[i]] + beta.Sow[sow[i]]
  for(t in (first[i]+1):last[i]){
    mu[i,t] <- phi[i]*y[i,t-1]
    y[i,t] ~dbern(mu[i,t])
    loglike[i,t] <- logdensity.bern(y[i,t], mu[i,t]) #for WAIC
  }
}
beta0 ~ dunif(-3,5)

for (t in 1:n.years){
  for (k in 1:n.sow){
    logit(mean.phi.female[t,k]) <- beta0 + beta.sex[1] + beta.year[t] + beta.Sow[k]
    logit(mean.phi.male[t,k]) <- beta0 + beta.sex[2]  + beta.year[t] + beta.Sow[k]
    male_42[t,k] <- pow(mean.phi.male[t,k], 42)
    female_42[t,k] <- pow(mean.phi.female[t,k], 42)
    }
  }
}
"
```

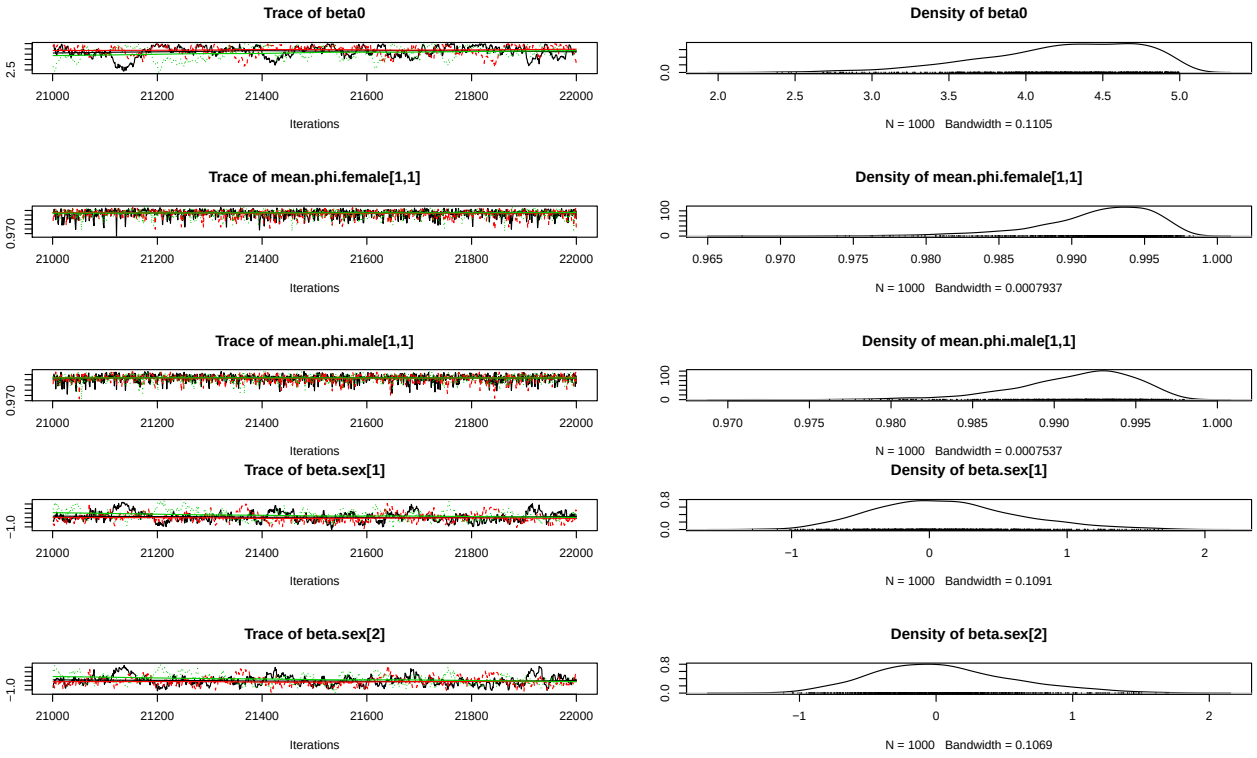
Note that pow(x, 42) in JAGS just means x^{42}.

When we give JAGS our year information, we want to make sure the years correspond to "1", "2" and "3".

```
jd <- list(n.ind = nrow(neos), y = surv.caps, first = first, last = last, sex = neos$sex,
    n.years = 3, year = neos$year - 2016, sow = neos$Sow_numeric, n.sow = n.sow)
ji <- function() {
    list(beta0 = 0.5, beta.sex = runif(2), beta.year = runif(3))
}
jp <- c("beta0", "mean.phi.female", "beta.year", "mean.phi.male", "male_42", "female_42",
    "beta.sex", "loglike", "beta.Sow")
library(runjags)
piggies.sexyear <- run.jags(model = modelstring.neos_sexyear, monitor = jp, data = jd,
    inits = ji, n.chains = 3, burnin = 20000, sample = 1000, adapt = 1000, method = "parallel",
    thin = 1)
## Warning: You attempted to start parallel chains without setting different PRNG
## for each chain, which is not recommended. Different .RNG.name values have been
## added to each set of initial values.
## 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 Nov 23 13:01:49 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: 2919
##    Unobserved stochastic nodes: 31
##    Total graph size: 4085
## . Reading parameter file inits1.txt
## . Initializing model
## . Adapting 1000
## -------------------------------------------------| 1000
## +++++++++++++++++++++++++++++++++++++++++++++++++ 100%
## Adaptation successful
## . Updating 20000
## -------------------------------------------------| 20000
## ************************************************* 100%
## . . . . . . . . . . Updating 1000
## -------------------------------------------------| 1000
## ************************************************* 100%
## . . . . Updating 0
## . Deleting model
## All chains have finished
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Note: Summary statistics were not produced as there are >50 monitored
## variables
## [To override this behaviour see ?add.summary and ?runjags.options]
## FALSEFinished running the simulation
```

Check a few results. Let's just look at results for individuals that have Sow 1 has their mom.

```
plot(piggies.sexyear$mcmc[, c("beta0", "mean.phi.female[1,1]", "mean.phi.male[1,1]",
    "beta.sex[1]", "beta.sex[2]"), ])
```




```
summary(piggies.sexyear$mcmc[, c("beta0", "mean.phi.female[1,1]", "mean.phi.male[1,1]",
    "male_42[1,1]", "male_42[2,1]", "male_42[3,1]", "female_42[1,1]", "female_42[2,1]",
    "female_42[3,1]", "beta.sex[1]", "beta.sex[2]", "beta.year[1]", "beta.year[2]",
    "beta.year[3]"), ])
##
## Iterations = 21001:22000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                          Mean       SD  Naive SE Time-series SE
## beta0                 4.23019 0.517127 9.441e-03      5.146e-02
## mean.phi.female[1,1]  0.99166 0.004129 7.539e-05      9.872e-05
## mean.phi.male[1,1]    0.99128 0.003736 6.821e-05      8.107e-05
## male_42[1,1]          0.70019 0.103422 1.888e-03      2.244e-03
## male_42[2,1]          0.52310 0.150446 2.747e-03      4.833e-03
## male_42[3,1]          0.67088 0.130226 2.378e-03      3.198e-03
## female_42[1,1]        0.71339 0.113260 2.068e-03      2.713e-03
## female_42[2,1]        0.54323 0.157490 2.875e-03      4.538e-03
## female_42[3,1]        0.68513 0.137429 2.509e-03      3.831e-03
## beta.sex[1]           0.12422 0.520024 9.494e-03      4.350e-02
## beta.sex[2]           0.05376 0.517068 9.440e-03      4.163e-02
## beta.year[1]          0.26084 0.292113 5.333e-03      1.189e-02
## beta.year[2]         -0.36442 0.305365 5.575e-03      1.361e-02
```

```
## beta.year[3]          0.16251 0.303533 5.542e-03      9.062e-03
##
## 2. Quantiles for each variable:
##
##                          2.5%      25%        50%     75%  97.5%
## beta0                  3.0269  3.92173  4.302e+00  4.6500 4.9528
## mean.phi.female[1,1]   0.9812  0.98969  9.925e-01  0.9947 0.9972
## mean.phi.male[1,1]     0.9820  0.98923  9.919e-01  0.9940 0.9967
## male_42[1,1]           0.4654  0.63447  7.118e-01  0.7751 0.8714
## male_42[2,1]           0.2012  0.42674  5.343e-01  0.6334 0.7771
## male_42[3,1]           0.3605  0.59205  6.883e-01  0.7639 0.8749
## female_42[1,1]         0.4497  0.64711  7.278e-01  0.7988 0.8871
## female_42[2,1]         0.2092  0.44032  5.565e-01  0.6607 0.8057
## female_42[3,1]         0.3638  0.60401  7.101e-01  0.7854 0.8903
## beta.sex[1]           -0.7700 -0.24786  8.532e-02  0.4360 1.2822
## beta.sex[2]           -0.8349 -0.30729 -3.302e-05  0.3628 1.1938
## beta.year[1]          -0.2657  0.06223  2.370e-01  0.4412 0.9046
## beta.year[2]          -1.0401 -0.55333 -3.321e-01 -0.1345 0.1204
## beta.year[3]          -0.3813 -0.03355  1.374e-01  0.3482 0.8213
```

So what do these results tell us? Firstly, if you were born in year 2, your survival chances are probably lower than piglets born in other years (based on the beta.year estimates). Secondly there is some evidence of higher survival for male piglets (beta.sex[2] is larger than beta.sex[1]) but not much of a difference between sexes.

Let's look at one last model - dealing with survival changing as piglets age.

Time Varying Survival

One thing about pigs is that they grow pretty fast. The day they are born, they're still pretty small and bad at surviving but by the end of the first month their survival skills have improved dramatically.

So how do we incorporate this into the model? If we had time-varying covariates that would be an easy way to change the model, but in this case we don't. So instead, we can "make" one to represent the age of the pig.

In theory we could make age a variable in our data frame and give the model the pig's age at any given time point. Or we could just do that *directly in the model* which is much faster. And feels more badass.

We do this by calculating current age of the pig as the current time period we are in, t, minus the first time we saw the pig (the day it was born). So if we are in the 25th time period but the piglet was born on the 10th day, the piglet would be age 15 days.

Let's try adding age and Sow effect into the same model.

```
modelstring.neos_SowAge = "
model {

sigmaSow ~ dunif(0, 2)\t\t# Random effect SD between 0 and 2
  tauSow <- 1 / (sigmaSow * sigmaSow)
for(k in 1:n.sow){
  beta.Sow[k] ~ dnorm(0, tauSow)
}

for (i in 1:n.ind){
  for(t in (first[i]+1):last[i]){
  logit(phi[i,t]) <- beta0 + beta.Sow[sow[i]] + beta.age*(t-first[i])
    mu[i,t] <- phi[i,t]*y[i,t-1]
    y[i,t] ~dbern(mu[i,t])
```

```
    loglike[i,t] <- logdensity.bern(y[i,t], mu[i,t]) #for WAIC
  }
}
beta0 ~ dunif(-6,6)
beta.age ~ dnorm(0,.5) #constrain to avoid difficulties with convergence

}
"
```

Not much changes from our sow model except that phi is now indexed by both i and t.

```
jd <- list(n.ind = nrow(neos), y = surv.caps, first = first, last = last, sow = neos$Sow_numeric,
    n.sow = n.sow)
ji <- function() {
    list(beta0 = 0.5, beta.Sow = runif(n.sow))
}
jp <- c("beta0", "loglike", "beta.Sow", "beta.age")
library(runjags)
piggies.SowAge <- run.jags(model = modelstring.neos_SowAge, monitor = jp, data = jd,
    inits = ji, n.chains = 3, burnin = 1000, sample = 2000, adapt = 1000, method = "parallel",
    thin = 1)
## Warning: You attempted to start parallel chains without setting different PRNG
## for each chain, which is not recommended. Different .RNG.name values have been
## added to each set of initial values.
## 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 Nov 23 13:04:14 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: 2919
##    Unobserved stochastic nodes: 26
##    Total graph size: 8125
## . Reading parameter file inits1.txt
## . Initializing model
## . Adapting 1000
## -------------------------------------------------| 1000
## +++++++++++++++++++++++++++++++++++++++++++++++++ 100%
## Adaptation successful
## . Updating 1000
## -------------------------------------------------| 1000
## ************************************************* 100%
## . . . . . Updating 2000
## -------------------------------------------------| 2000
## ************************************************* 100%
## . . . . Updating 0
## . Deleting model
## All chains have finished
```

```
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Note: Summary statistics were not produced as there are >50 monitored
## variables
## [To override this behaviour see ?add.summary and ?runjags.options]
## FALSEFinished running the simulation
```

Check our output as always! Here's an additional trick - if you want to plot all the sows but don't want to write out every single indexed parameter, you can use "grep" to list all the variables following a given pattern:

```
varnames(piggies.SowAge$mcmc)[grep("beta.Sow\\[", varnames(piggies.SowAge$mcmc))]
##  [1] "beta.Sow[1]"  "beta.Sow[2]"  "beta.Sow[3]"  "beta.Sow[4]"  "beta.Sow[5]"
##  [6] "beta.Sow[6]"  "beta.Sow[7]"  "beta.Sow[8]"  "beta.Sow[9]"  "beta.Sow[10]"
## [11] "beta.Sow[11]" "beta.Sow[12]" "beta.Sow[13]" "beta.Sow[14]" "beta.Sow[15]"
## [16] "beta.Sow[16]" "beta.Sow[17]" "beta.Sow[18]" "beta.Sow[19]" "beta.Sow[20]"
## [21] "beta.Sow[21]" "beta.Sow[22]" "beta.Sow[23]"
```

Here we can put that to use:

```
summary(piggies.SowAge$mcmc[, c("beta0", varnames(piggies.SowAge$mcmc)[grep("beta.Sow\\[",
    varnames(piggies.SowAge$mcmc))], "beta.age"), ])
##
## Iterations = 2001:4000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 2000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                  Mean      SD  Naive SE Time-series SE
## beta0         3.53303 0.46557 0.0060104      0.0268988
## beta.Sow[1]   0.72798 0.85907 0.0110905      0.0170940
## beta.Sow[2]   1.51828 1.09382 0.0141211      0.0255537
## beta.Sow[3]   0.37562 0.89831 0.0115971      0.0195599
## beta.Sow[4]   0.66943 1.23222 0.0159079      0.0218750
## beta.Sow[5]   1.40813 1.10320 0.0142422      0.0239284
## beta.Sow[6]   0.68332 0.88462 0.0114204      0.0196473
## beta.Sow[7]   0.44946 0.90150 0.0116383      0.0221710
## beta.Sow[8]  -1.67878 0.64110 0.0082766      0.0248563
## beta.Sow[9]   0.26658 0.92999 0.0120061      0.0194672
## beta.Sow[10] -0.17214 0.95140 0.0122825      0.0189800
## beta.Sow[11]  0.02843 0.92467 0.0119375      0.0178437
## beta.Sow[12] -1.07128 0.65550 0.0084625      0.0216745
## beta.Sow[13]  1.13712 1.12154 0.0144791      0.0214699
## beta.Sow[14] -1.62011 0.77407 0.0099932      0.0246962
## beta.Sow[15] -2.05395 0.73963 0.0095485      0.0284788
## beta.Sow[16] -2.19123 0.69171 0.0089300      0.0275134
## beta.Sow[17] -1.45282 0.63631 0.0082147      0.0261390
## beta.Sow[18]  0.44221 0.87641 0.0113144      0.0170450
## beta.Sow[19]  1.38922 1.11312 0.0143703      0.0209949
## beta.Sow[20]  0.22951 0.93446 0.0120638      0.0193006
## beta.Sow[21]  1.39966 1.11826 0.0144367      0.0213117
## beta.Sow[22] -1.78968 0.96680 0.0124813      0.0269699
## beta.Sow[23]  0.96485 1.17558 0.0151767      0.0229178
```

```
## beta.age      0.08948 0.02417 0.0003121      0.0007857
##
## 2. Quantiles for each variable:
##
##                 2.5%      25%      50%     75%   97.5%
## beta0         2.63408  3.21814  3.52693  3.8395  4.4574
## beta.Sow[1]  -0.76000  0.12710  0.65133  1.2762  2.5790
## beta.Sow[2]  -0.31679  0.74056  1.40972  2.1662  4.0266
## beta.Sow[3]  -1.20025 -0.25217  0.31870  0.9303  2.3001
## beta.Sow[4]  -1.50879 -0.19527  0.57358  1.4187  3.3241
## beta.Sow[5]  -0.44289  0.64100  1.29203  2.0610  3.9040
## beta.Sow[6]  -0.88222  0.05943  0.62949  1.2390  2.6066
## beta.Sow[7]  -1.14857 -0.16940  0.39942  1.0147  2.3953
## beta.Sow[8]  -2.91582 -2.12197 -1.68540 -1.2454 -0.3865
## beta.Sow[9]  -1.38062 -0.38161  0.20436  0.8428  2.2504
## beta.Sow[10] -1.91373 -0.84934 -0.22252  0.4304  1.8477
## beta.Sow[11] -1.60084 -0.62857 -0.02136  0.6383  1.9364
## beta.Sow[12] -2.29697 -1.52360 -1.09033 -0.6428  0.2830
## beta.Sow[13] -0.84193  0.36381  1.05496  1.8325  3.5915
## beta.Sow[14] -3.05503 -2.15249 -1.65661 -1.1111 -0.0456
## beta.Sow[15] -3.44629 -2.55654 -2.06315 -1.5682 -0.5596
## beta.Sow[16] -3.50697 -2.66859 -2.21510 -1.7426 -0.8135
## beta.Sow[17] -2.63772 -1.89231 -1.46405 -1.0219 -0.1878
## beta.Sow[18] -1.09664 -0.15705  0.36827  0.9855  2.3351
## beta.Sow[19] -0.46098  0.59981  1.30453  2.0624  3.8435
## beta.Sow[20] -1.41201 -0.41135  0.14749  0.8086  2.2689
## beta.Sow[21] -0.44659  0.61223  1.29547  2.0776  3.9175
## beta.Sow[22] -3.61528 -2.44298 -1.82165 -1.1563  0.1680
## beta.Sow[23] -1.10691  0.15605  0.85541  1.6777  3.5585
## beta.age      0.04575  0.07258  0.08805  0.1051  0.1409
```

These results suggest that not only does survival vary with Sow, it also appears to increase as the piglet gets older. This matches what we already know/suspect about piglets, so this shouldn't come as a surprise.

Now let's use that loglikelihood calculation we added to our code to rank our models.

WAIC Table

Fair warning, I am absurdly proud of this WAIC table code. It's just so efficient! First step, we need to write a function to turn our beautiful loglikelihoods that we calculated into WAIC.

```
calc.waic <- function(x) {
    vars <- grep("loglike", colnames(x$mcmc[[1]]))
    # find the output that relates to loglike
    like <- as.matrix(x$mcmc[, vars, ])
    fbar <- colMeans(like)  #mean log-likelihood
    Pw <- sum(apply(like, 2, var))  #mean variance in log-likelihood
    WAIC <- -2 * sum(fbar) + 2 * Pw
    return(WAIC)
}
```

This function can be used on individual models if we want.

```
calc.waic(piggies.null)
## [1] 399.88
```

But if you have a bunch of models, that's a pain. So, here comes the part of the code that I'm proud of.

We can ask R for all the models that follow a similar name that are listed in our R environment. This code is actually storing the objects themselves as list in a new object, rather than just grabbing the names of the models.

```
mymodels <- mget(ls()[grep("piggies", ls())])  #grabs all models in your environment
mymodels
## $piggies.null
##
## JAGS model with 6000 samples (chains = 3; adapt+burnin = 2000)
##
## Full summary statistics have not been pre-calculated - use either the summary method or add.summary
##
##
## $piggies.sexyear
##
## JAGS model with 3000 samples (chains = 3; adapt+burnin = 21000)
##
## Full summary statistics have not been pre-calculated - use either the summary method or add.summary
##
##
## $piggies.Sow
##
## JAGS model with 6000 samples (chains = 3; adapt+burnin = 2000)
##
## Full summary statistics have not been pre-calculated - use either the summary method or add.summary
##
##
## $piggies.SowAge
##
## JAGS model with 6000 samples (chains = 3; adapt+burnin = 2000)
##
## Full summary statistics have not been pre-calculated - use either the summary method or add.summary
```

Now we can use our WAIC function on all of these models and output a table.

```
# empty data frame
WAIC <- data.frame(modname = ls()[grep("piggies", ls())], WAIC = rep(NA, length(mymodels)))
# run the WAIC code for each model
for (i in 1:length(mymodels)) {
    WAIC[i, 2] <- calc.waic(mymodels[[i]])
}

# model weights table
WAIC$deltaWAIC <- WAIC$WAIC - min(WAIC$WAIC)
WAIC$rel_like <- exp(-0.5 * WAIC$deltaWAIC)
WAIC$weight <- WAIC$rel_like/sum(WAIC$rel_like)
```

```
WAIC[order(-WAIC$weight), ]
```

	modname	WAIC	deltaWAIC	rel_like	weight
4	piggies.SowAge	332.30	0.00	1	1
2	piggies.sexyear	372.48	40.18	0	0
3	piggies.Sow	375.33	43.04	0	0

	modname	WAIC	deltaWAIC	rel_like	weight
1	piggies.null	399.88	67.58	0	0

Looks like all our weight is on the model SowAge! Of course, we didn't test all combinations of variables so we only know it's the best model of the models we tested.

Graphing Survival Curves

In the case of this study, the researchers were interested in the probability of an individual surviving for 42 days. A nice way to report the findings is a survival curve. We calculate the daily probability for individuals in our study, but the differences in daily probabilities don't always stand out without a visual.

Survival curves are just the cumulative probability of an animal surviving a certain number of days. When survival probability is the same at each time step, it's just phi^{time}. When it changes by time, it's just the product of each daily probability.

First, we need to grab the parameters needed to calculate the daily survival for piglets from each Sow. We'll use the quantiles part of the output.

```
variables <- summary(piggies.SowAge$mcmc[, c("beta0", varnames(piggies.SowAge$mcmc)[grep("beta.Sow\\[",
    varnames(piggies.SowAge$mcmc))], "beta.age"), ])$quantiles
head(variables, n = 3)
##                  2.5%       25%       50%      75%    97.5%
## beta0        2.6340805 3.2181425 3.5269350 3.839497 4.457351
## beta.Sow[1] -0.7600048 0.1270965 0.6513275 1.276185 2.578994
## beta.Sow[2] -0.3167922 0.7405598 1.4097150 2.166192 4.026568
```

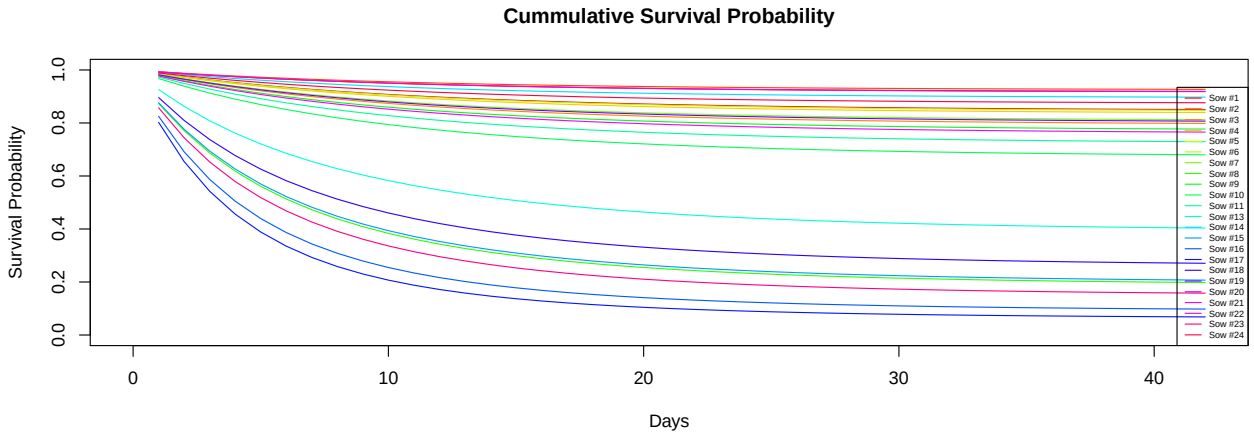
Now, we can stick these values into an equation. Our output will have one row for each sow and one column for each daily survival probability. We'll calculate out to 42 days.

```
daily.surv <- cum.surv <- matrix(NA, nrow = n.sow, ncol = 42)
for (i in 1:n.sow) {
    # plogis(beta.0 + beta.Sow + beta.age*age)
    daily.surv[i, ] <- plogis(variables[1, 3] + variables[i + 1, 3] + variables[25,
        3] * 1:42)
    cum.surv[i, ] <- cumprod(daily.surv[i, ])
}
```

The natural impulse might be to graph the daily probabilities straight up, but that only tells you how daily survival is changing over time, not the probability of surviving a certain length of time.

For base R:

```
plot(1:42, cum.surv[1, ], type = "l", col = rainbow(n.sow)[1], xlim = c(0, 42), ylim = c(0,
    1), main = "Cummulative Survival Probability", xlab = "Days", ylab = "Survival Probability")
for (i in 2:n.sow) {
    lines(1:42, cum.surv[i, ], type = "l", col = rainbow(n.sow)[i])
}
legend("bottomright", paste("Sow #", c(1:11, 13:24), sep = ""), lty = 1, col = rainbow(n.sow),
    cex = 0.5)
```

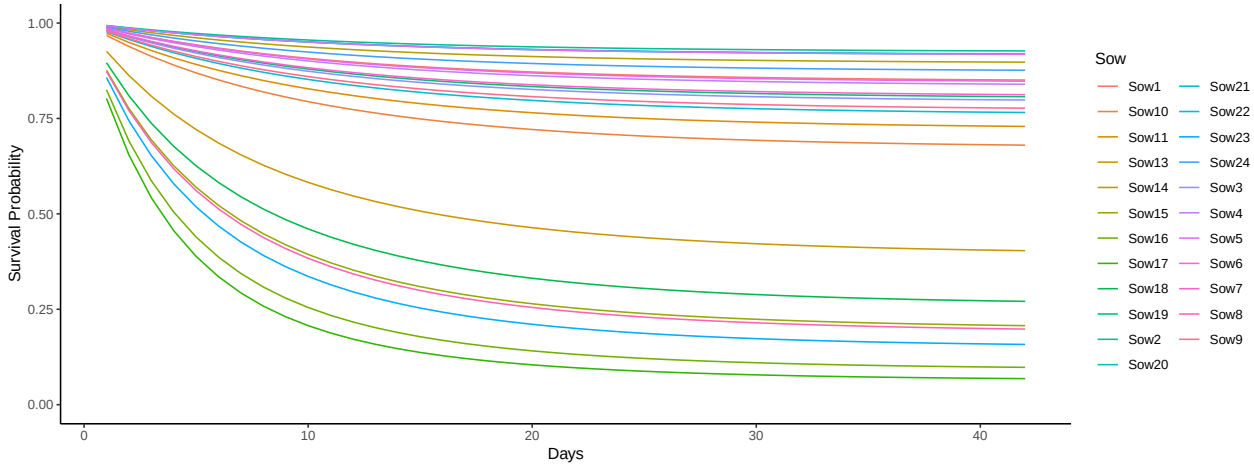



```
# Sow 12 didn't have any babies in our data set
```

For ggplot2, we have to change the format of our data into long format, which is easy to do with the pivot_longer function.

```
library(ggplot2)
library(tidyr)
cum.surv <- as.data.frame(cum.surv)
cum.surv$Sow = paste("Sow", c(1:11, 13:24), sep = "")
long_surv <- pivot_longer(cum.surv, cols = starts_with("V"))
long_surv$age <- rep(1:42, n.sow)

print(ggplot(long_surv, aes(x = age, y = value, col = Sow, group = Sow)) + geom_line() +
    theme_classic() + ylim(0, 1) + xlab("Days") + ylab("Survival Probability"))
```



Awesome!

Known Fate Models in NIMBLE

Running known fate models in NIMBLE is actually not that different from running them in JAGS. BUT you can do run more of them much more efficiently. For one thing, because you can run a model with multiple datasets without having to re-compile the model, you can write just one big model with all your covariates and then turn the covariates on and off for each model run. This is a HUGE time saver. Technically you could do this in JAGS to, but it wouldn't save you any time except for the physical typing part.

So first, let's build our big model with all the covariates in it. Remember that in NIMBLE we can ask it to

calculate WAIC for us, plus we can use sd or precision in our normal distributions!

```
piggies.all <- nimbleCode({

    # sow effects
    sigmaSow ~ dunif(0, 0.5)
    for (k in 1:n.sow) {
        beta.Sow[k] ~ dnorm(0, sd = sigmaSow)
    }

    # sex effects
    beta.sex[1] ~ dnorm(0, sd = 0.5)  #female
    beta.sex[2] ~ dnorm(0, sd = 0.5)  #male


    sigmayear ~ dunif(0, 0.5)  # Random effect SD

    for (t in 1:n.years) {
        beta.year[t] ~ dnorm(0, sd = sigmayear)
    }


    for (i in 1:n.ind) {
        for (t in (first[i] + 1):last[i]) {
            logit(phi[i, t]) <- beta0 + beta.Sow[sow[i]] + beta.age * (t - first[i]) +
                beta.sex[sex[i]] + beta.year[year[i]] + beta.mass * mass[i]
            mu[i, t] <- phi[i, t] * y[i, t - 1]
            y[i, t] ~ dbern(mu[i, t])
        }
    }
    beta0 ~ dunif(-6, 6)
    beta.age ~ dnorm(0, 0.5)  #constrain to avoid difficulties with convergence
    beta.mass ~ dnorm(0, 0.5)  #constrain to avoid difficulties with convergence
})
```

Now we can add in the "switches" which we will set to 1 to "turn on" the covariate or 0 to "turn off". On will be given to NIMBLE as data in the form of a vector of 5 numbers (0 and 1's).

```
piggies.all <- nimbleCode({

    # sow effects
    sigmaSow ~ dunif(0, 2)
    for (k in 1:n.sow) {
        beta.Sow[k] ~ dnorm(0, sd = sigmaSow)
    }

    # sex effects
    beta.sex[1] ~ dnorm(0, sd = 0.5)  #female
    beta.sex[2] ~ dnorm(0, sd = 0.5)  #male


    sigmayear ~ dunif(0, 2)  # Random effect SD

    for (t in 1:n.years) {
        beta.year[t] ~ dnorm(0, sd = sigmayear)
```

```
    }


    for (i in 1:n.ind) {
        for (t in (first[i] + 1):last[i]) {
            logit(phi[i, t]) <- beta0 + beta.Sow[sow[i]] * on[1] + beta.age * (t -
                first[i]) * on[2] + beta.sex[sex[i]] * on[3] + beta.year[year[i]] *
                on[4] + beta.mass * mass[i] * on[5]

            mu[i, t] <- phi[i, t] * y[i, t - 1]
            y[i, t] ~ dbern(mu[i, t])
        }
    }
    beta0 ~ dunif(-6, 6)
    beta.age ~ dnorm(0, sd = 0.25)  #constrain to avoid difficulties with convergence
    beta.mass ~ dnorm(0, sd = 0.25)  #constrain to avoid difficulties with convergence
})
```

We could also write derived params into our model and just use the same switches that we use to turn covariates on and off, but I'll skip that for now.

Next we can get our data, inits and params ready to go to run our model. First we'll run the null model but everything except the "on" vector in data will be the same for all runs.

Intercept Only

```
n.params <- c("beta0", "beta.Sow", "beta.age", "beta.sex", "beta.year", "beta.mass")
n.constants <- list(n.ind = nrow(neos), first = first, last = last, sow = neos$Sow_numeric,
    n.sow = n.sow, mass = neos$mass_s, year = neos$year - 2016, sex = neos$sex, n.years = 3)
n.data <- list(y = surv.caps, on = c(0, 0, 0, 0, 0))
n.inits <- list(beta0 = runif(1), beta.age = runif(1), beta.mass = runif(1), beta.Sow = runif(n.sow),
    beta.sex = runif(2), sigmayear = runif(1), sigmaSow = runif(1))
```

Time to start our model runs.

```
preppigs <- nimbleModel(code = piggies.all, constants = n.constants, data = n.data,
    inits = n.inits)
## defining model...
## building model...
## setting data and initial values...
## running calculate on model (any error reports that follow may simply reflect missing values in model
## checking model sizes and dimensions... This model is not fully initialized. This is not an error. To
## model building finished.
preppigs$initializeInfo()
## Missing values (NAs) or non-finite values were found in model variables: beta.year, phi, mu, y. This
```

Hmm, we seem to be missing some values that we'll want to initialize. First we'll have NIMBLE find values for beta.year, then we'll have it follow through with those values to initialize phi and mu. Note that y will always have some NA's - those are all the times the piglets weren't born or we weren't following them around.

```
preppigs$simulate("beta.year")
preppigs$calculate("phi")
## [1] 0
preppigs$calculate("mu")
## [1] 0
```

Notice if we run initializeinfo() again, we'll still have NA's. This is because phi and mu are also unknown for large quantities of time in our model. That's okay! We don't need to know them at every step because they aren't relevant.

Onwards to the other parts of the model. Make sure to add "enableWAIC = T"

```
mcmcpigs <- configureMCMC(preppigs, monitors = n.params, print = T, enableWAIC = T)
## ===== Monitors =====
## thin = 1: beta0, beta.Sow, beta.age, beta.sex, beta.year, beta.mass
## ===== Samplers =====
## RW sampler (33)
##   - sigmaSow
##   - beta.sex[]  (2 elements)
##   - sigmayear
##   - beta0
##   - beta.age
##   - beta.mass
##   - beta.Sow[]  (23 elements)
##   - beta.year[]  (3 elements)
pigsMCMC <- buildMCMC(mcmcpigs)
## Monitored nodes are valid for WAIC
Cmodel <- compileNimble(preppigs)
## compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilation details.
## compilation finished.
Comppigs <- compileNimble(pigsMCMC, project = preppigs)
## compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilation details.
## compilation finished.
Comppigs$run(niter = 1000, nburnin = 500)
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
## NULL
```

In a minute we'll run this in parallel, but for now we're just doing some test chains to make sure everything is working. Let see what we get out:

```
tail(as.mcmc(as.matrix(Comppigs$mvSamples)), n = 5)
## Markov Chain Monte Carlo (MCMC) output:
## Start = 495
## End = 500
## Thinning interval = 1
##      beta.Sow[1] beta.Sow[2] beta.Sow[3] beta.Sow[4] beta.Sow[5] beta.Sow[6]
## [1,]   -2.689311   -1.118935  -3.0914371   0.9254826   1.2551918   0.1059226
## [2,]   -2.801542   -1.118935  -1.0425756   0.9254826  -0.3852381  -3.2405824
## [3,]   -0.375331   -1.118935  -1.0425756   4.9391946  -1.3394595  -1.4568375
## [4,]   -2.393903   -1.118935  -1.0425756   4.9391946  -1.3394595  -1.4568375
## [5,]   -2.393903   -1.118935  -0.1740542   4.7567003  -1.2916585  -1.7893079
## [6,]   -2.393903    1.515983  -0.1740542   4.3610186  -1.2916585  -1.0177735
##      beta.Sow[7] beta.Sow[8] beta.Sow[9] beta.Sow[10] beta.Sow[11] beta.Sow[12]
## [1,]   0.3730852  -0.8326902   0.4034907     2.827510   -0.5359174  -0.06595573
## [2,]   0.3730852  -0.8326902   0.4034907     4.469710   -0.5359174  -0.06595573
## [3,]   0.3730852   3.2833312   0.4034907     5.358794   -0.5359174   1.08705130
## [4,]   0.3730852   3.2259102   0.4034907     5.051949   -1.6351182   1.08705130
## [5,]   0.3730852   3.2259102   0.4034907     1.415045    3.0397391   1.08705130
## [6,]   0.3730852  -1.2731318  -1.8820248     1.415045    1.5937773   3.26339308
##      beta.Sow[13] beta.Sow[14] beta.Sow[15] beta.Sow[16] beta.Sow[17]
## [1,]   -4.0337767   -0.6957509   -1.9906508    0.4838592    0.2389554
```

```
## [2,]   -0.5829797   -0.9805195    1.1823651    0.4838592    0.1465384
## [3,]   -0.5829797    3.1784379   -0.9901907    0.4838592    0.1465384
## [4,]    0.4144848    1.1160529   -0.9901907    0.7943201    0.1465384
## [5,]   -0.1532837    1.1160529   -2.5808844    0.7137382   -1.9034575
## [6,]   -2.9127908    2.2352345   -3.9062504   -2.0066964    2.4310402
##      beta.Sow[18] beta.Sow[19] beta.Sow[20] beta.Sow[21] beta.Sow[22]
## [1,]    0.7410182   -1.7986809   -0.6965713     3.159568     3.379941
## [2,]   -3.0023650   -1.4135870   -0.6965713     3.159568     3.379941
## [3,]   -2.7938309    0.3709584   -0.6965713     4.145948     3.379941
## [4,]   -0.7853227    0.3709584   -0.6965713     4.226688     2.580280
## [5,]   -0.7853227   -2.4897307    1.6165534     4.226688     2.580280
## [6,]   -1.0218338    0.4178166    1.6165534     4.051625     2.580280
##      beta.Sow[23]    beta.age   beta.mass beta.sex[1] beta.sex[2] beta.year[1]
## [1,]    0.6484745 -0.07805997 -0.25103916  -0.6575321  0.02409901    -1.592899
## [2,]    0.6484745 -0.07805997 -0.03087306   0.1930604  0.02409901    -1.592899
## [3,]    0.6484745 -0.07805997  0.43463048   0.6208008  0.02409901    -1.592899
## [4,]   -1.4023655 -0.07805997  0.62292685   0.6208008  0.02409901    -1.592899
## [5,]    0.6993560 -0.07805997  0.61599519   0.6208008  0.02409901    -1.592899
## [6,]   -0.4647299 -0.07805997  0.45214350   0.6208008  0.52317537     1.247079
##      beta.year[2] beta.year[3]    beta0
## [1,]   -2.3564180    1.0337937 4.618406
## [2,]   -1.4396414    2.6965833 4.546475
## [3,]   -0.2879724    2.6965833 4.487932
## [4,]   -0.8579924    2.6965833 4.542996
## [5,]   -0.3855530    0.9740401 4.492171
## [6,]   -0.3855530    0.9740401 4.492171
```

A lot of output! But remember because we "turned off" all the covariates, the only relevant one is beta0.

To get our WAIC out, we can use this method:

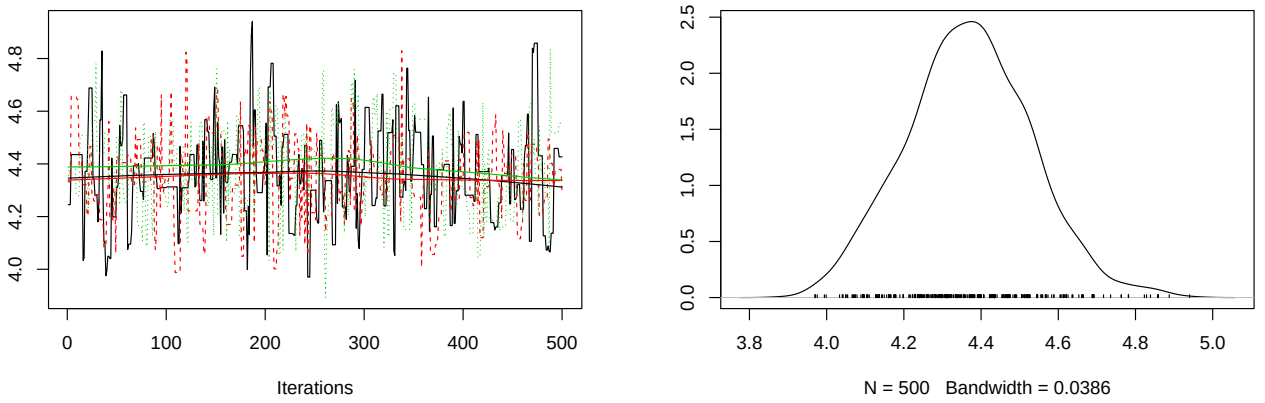
```
Comppigs$calculateWAIC()
## [1] 398.6106
```

Excellent. Let's try a full run of this model now with 3 chains.

```
library(parallel)
cl <- makeCluster(3)
clusterExport(cl = cl, varlist = c("n.constants", "n.data", "n.inits", "n.params",
    "piggies.all"))
nimblepigs.null <- clusterEvalQ(cl = cl, {
    library(nimble)
    library(coda)
    preppigs <- nimbleModel(code = piggies.all, constants = n.constants, data = n.data,
        inits = n.inits)
    preppigs$simulate("beta.year")
    preppigs$calculate("phi")
    preppigs$calculate("mu")
    mcmcpigs <- configureMCMC(preppigs, monitors = n.params, enableWAIC = T)
    pigsMCMC <- buildMCMC(mcmcpigs)
    Cmodel <- compileNimble(preppigs)
    Comppigs <- compileNimble(pigsMCMC, project = preppigs)
    Comppigs$run(niter = 1000, nburnin = 500)
    return(list(outs = as.mcmc(as.matrix(Comppigs$mvSamples)), waic = Comppigs$calculateWAIC()))
})
```

Let's check a few of our results

```
null.outputs <- mcmc.list(nimblepigs.null[[1]]$outs, nimblepigs.null[[2]]$outs, nimblepigs.null[[3]]$ou
plot(null.outputs[, "beta0", ])
```




```
summary(null.outputs[, "beta0"])
##
## Iterations = 1:500
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##       4.368278       0.161471       0.004169       0.007500
##
## 2. Quantiles for each variable:
##
##  2.5%   25%   50%   75% 97.5%
## 4.056 4.263 4.371 4.473 4.688
```

Awesome! Results look about the same for our null model as they did in JAGS. We will grab out WAIC values at the end, much like we did in JAGS.

Random Effects

Our next model is one that includes the Sow and the birth mass of the piglet. These are the 1st and 5th "switches" in our model.

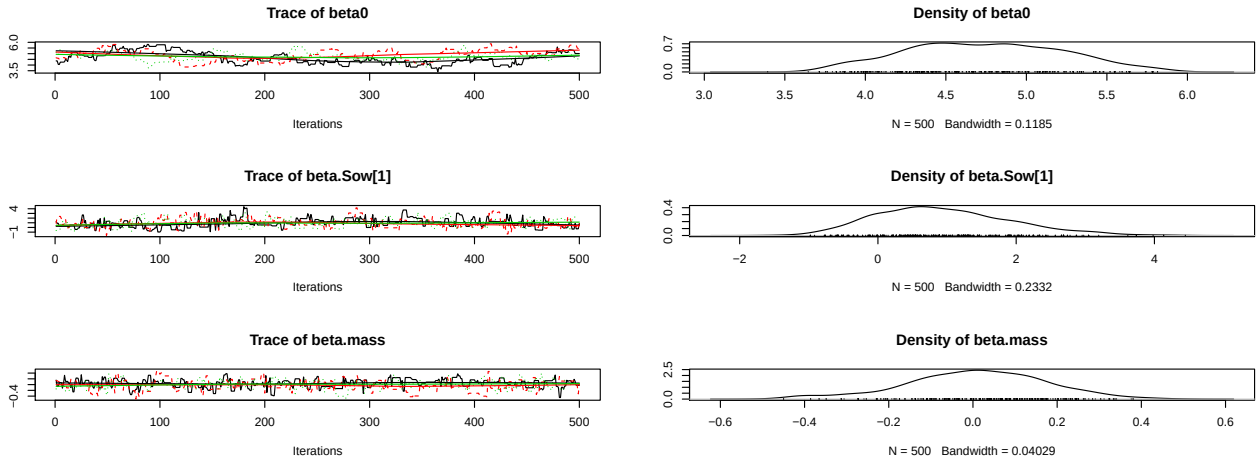
Without closing the cluster we made for the null model, we can just change the value of the switches and re run the model, only this time it will be running our random effects model! Note that we don't need to rerun the "nimbleModel" part of our code, we can just select the data in our "preppigs" object and change our values directly.

```
nimblepigs.Sow <- clusterEvalQ(cl = cl, {
    preppigs$on <- c(1, 0, 0, 0, 1)
    mcmcpigs <- configureMCMC(preppigs, monitors = n.params, enableWAIC = T)
    pigsMCMC <- buildMCMC(mcmcpigs)
    Cmodel <- compileNimble(preppigs)
    Comppigs <- compileNimble(pigsMCMC, project = preppigs)
    Comppigs$run(niter = 1000, nburnin = 500)
```

```
    return(list(outs = as.mcmc(as.matrix(Comppigs$mvSamples)), waic = Comppigs$calculateWAIC()))
})
```

Excellent, let's evaluate our results.

```
Sow.outputs <- mcmc.list(nimblepigs.Sow[[1]]$outs, nimblepigs.Sow[[2]]$outs, nimblepigs.Sow[[3]]$outs)
plot(Sow.outputs[, c("beta0", "beta.Sow[1]", "beta.mass"), ])
```




```
summary(Sow.outputs[, c("beta0", "beta.Sow[1]", "beta.Sow[2]", "beta.mass"), ])
##
## Iterations = 1:500
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                 Mean     SD Naive SE Time-series SE
## beta0       4.741009 0.4826 0.012460        0.09587
## beta.Sow[1] 0.946865 0.9680 0.024993        0.08466
## beta.Sow[2] 1.817897 1.1944 0.030840        0.07399
## beta.mass   0.006115 0.1667 0.004304        0.01007
##
## 2. Quantiles for each variable:
##
##                2.5%      25%      50%   75%  97.5%
## beta0       3.86263  4.38246 4.741113 5.117 5.6916
## beta.Sow[1] -0.65535  0.26538 0.859776 1.538 3.0973
## beta.Sow[2] -0.07842  0.94346 1.666419 2.591 4.6634
## beta.mass   -0.38842 -0.09687 0.009137 0.123 0.3172
```

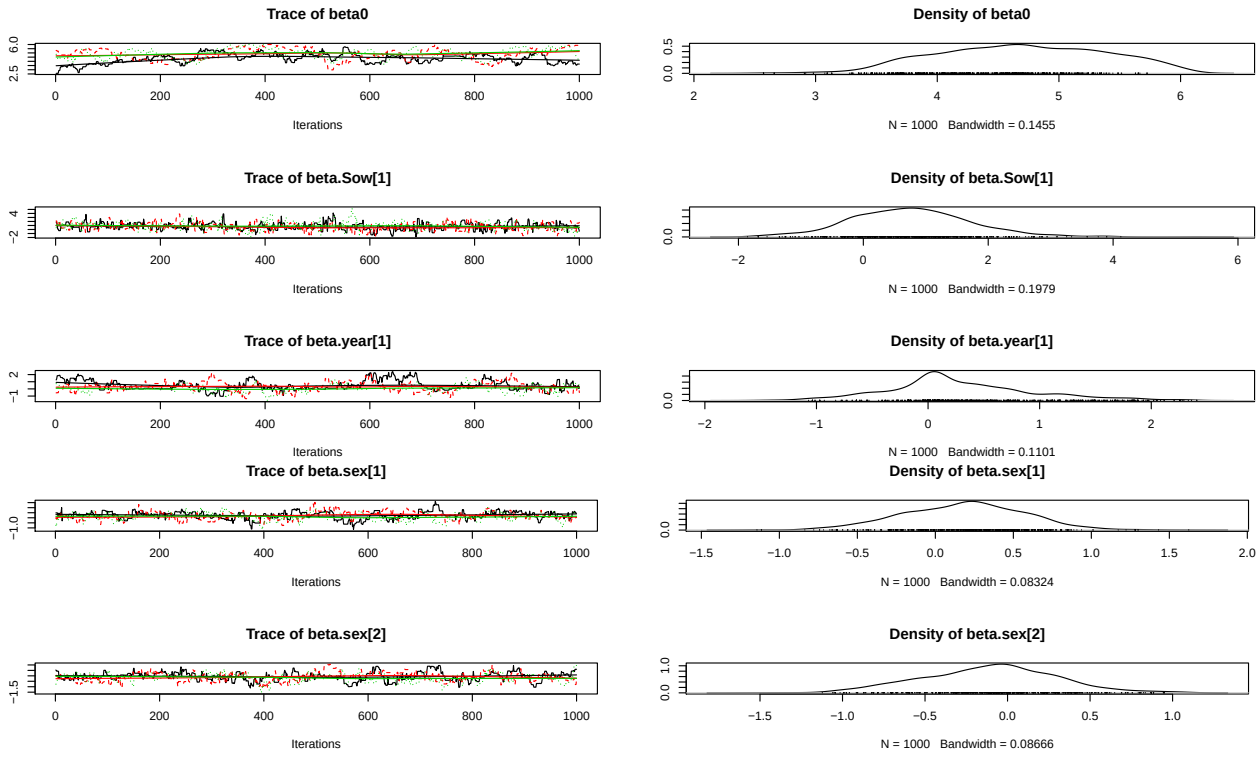
No surprise, the results are basically the same as we got from JAGS.

Random + Fixed Effects

Next comes our year model, which has sex, sow and year in it but does not have mass. This means we need to turn on switches 1, 3, and 4 but turn off all the others.

```
nimblepigs.sexyear <- clusterEvalQ(cl = cl, {
    preppigs$on <- c(1, 0, 1, 1, 0)
    mcmcpigs <- configureMCMC(preppigs, monitors = n.params, enableWAIC = T)
    pigsMCMC <- buildMCMC(mcmcpigs)
    Cmodel <- compileNimble(preppigs)
    Comppigs <- compileNimble(pigsMCMC, project = preppigs)
    Comppigs$run(niter = 3000, nburnin = 2000)
    return(list(outs = as.mcmc(as.matrix(Comppigs$mvSamples)), waic = Comppigs$calculateWAIC()))
})
```

```
sexyear.outputs <- mcmc.list(nimblepigs.sexyear[[1]]$outs, nimblepigs.sexyear[[2]]$outs,
    nimblepigs.sexyear[[3]]$outs)
plot(sexyear.outputs[, c("beta0", "beta.Sow[1]", "beta.year[1]", "beta.sex[1]", "beta.sex[2]"),
    ])
```




```
summary(sexyear.outputs[, c("beta0", "beta.Sow[1]", "beta.year[1]", "beta.year[2]",
    "beta.year[3]", "beta.sex[1]", "beta.sex[2]"), ])
##
## Iterations = 1:1000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                  Mean     SD Naive SE Time-series SE
## beta0          4.6915 0.6805 0.012424        0.10615
## beta.Sow[1]    0.7577 0.9569 0.017470        0.04403
## beta.year[1]   0.2824 0.6500 0.011868        0.06880
```

```
## beta.year[2] -0.6306 0.7808 0.014255         0.11868
## beta.year[3]  0.3371 0.7399 0.013509         0.06424
## beta.sex[1]   0.1779 0.3894 0.007110         0.03437
## beta.sex[2]  -0.1128 0.4054 0.007402         0.04052
##
## 2. Quantiles for each variable:
##
##                  2.5%      25%     50%      75%  97.5%
## beta0          3.4569  4.19816  4.6944  5.22282 5.9013
## beta.Sow[1]   -1.1098  0.10381  0.7330  1.34469 2.7602
## beta.year[1] -0.9159 -0.05680  0.1645  0.63365 1.8241
## beta.year[2] -2.1403 -1.15968 -0.5115 -0.01069 0.6472
## beta.year[3] -1.0775 -0.08128  0.2046  0.77013 1.9532
## beta.sex[1]  -0.6076 -0.08472  0.2015  0.44116 0.9367
## beta.sex[2]  -0.9093 -0.38827 -0.0914  0.16405 0.6860
```

We could probably have done with a few more iterations to make sure we don't introduce error into our estimates from the MCMC process, but the results are essentially the same from JAGS.

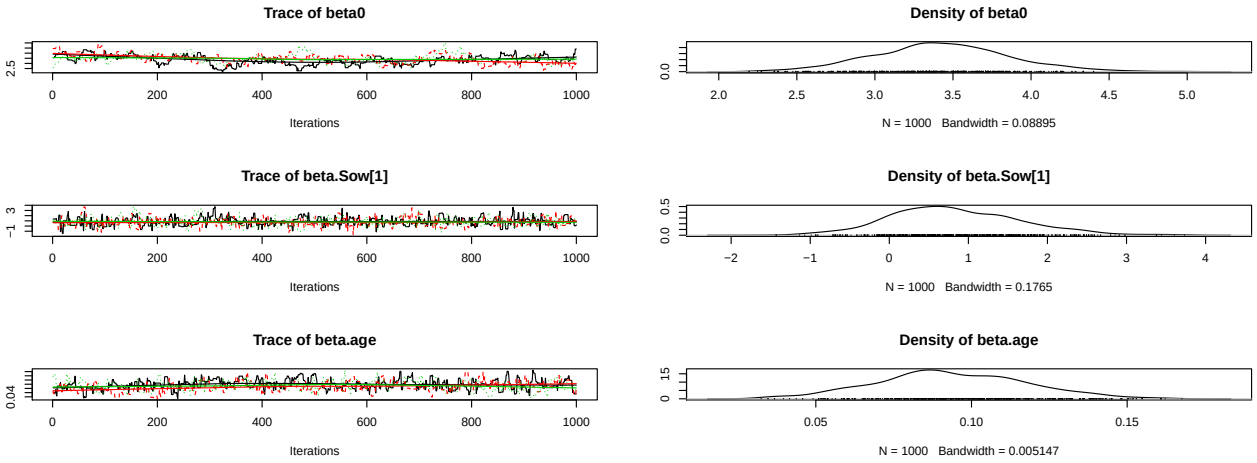
Finally, let's run the Time Varying model

Time Varying Survival

This model only includes an intercept, sow and age.

```
nimblepigs.SowAge <- clusterEvalQ(cl = cl, {
    preppigs$on <- c(1, 1, 0, 0, 0)
    mcmcpigs <- configureMCMC(preppigs, monitors = n.params, enableWAIC = T)
    pigsMCMC <- buildMCMC(mcmcpigs)
    Cmodel <- compileNimble(preppigs)
    Comppigs <- compileNimble(pigsMCMC, project = preppigs)
    Comppigs$run(niter = 3000, nburnin = 2000)
    return(list(outs = as.mcmc(as.matrix(Comppigs$mvSamples)), waic = Comppigs$calculateWAIC()))
})
```

```
sowage.outputs <- mcmc.list(nimblepigs.SowAge[[1]]$outs, nimblepigs.SowAge[[2]]$outs,
    nimblepigs.SowAge[[3]]$outs)
plot(sowage.outputs[, c("beta0", "beta.Sow[1]", "beta.age"), ])
```



```
summary(sowage.outputs[, c("beta0", "beta.Sow[1]", "beta.Sow[2]", "beta.age"), ])
##
## Iterations = 1:1000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                Mean      SD  Naive SE Time-series SE
## beta0       3.42293 0.43589 0.0079581       0.047656
## beta.Sow[1] 0.82735 0.82718 0.0151021       0.035673
## beta.Sow[2] 1.52277 1.02731 0.0187560       0.042908
## beta.age    0.09341 0.02408 0.0004397       0.001535
##
## 2. Quantiles for each variable:
##
##                2.5%     25%     50%    75%  97.5%
## beta0       2.57671 3.14401 3.41989 3.7017 4.3276
## beta.Sow[1] -0.68570 0.25850 0.75583 1.3649 2.5505
## beta.Sow[2] -0.14983 0.80424 1.38185 2.1629 3.7939
## beta.age     0.04752 0.07781 0.09153 0.1104 0.1407
```

Time for the WAIC table!

WAIC Table

We can use similar code to collect our model WAICs, but this time the WAIC scores themselves are calculated which saves us some time. This is not as clean a code as the one we can use for JAGS, but it gets the job done.

```
nim.models <- mget(ls()[grep("nimblepigs", ls())])  #grabs all models in your environment!
```

```
WAIC_nim <- data.frame(modname = ls()[grep("nimblepigs", ls())], WAIC = rep(NA, length(nim.models)))

for (i in 1:length(nim.models)) {
    WAIC_nim[i, 2] <- mean(unlist(nim.models[[i]], F)[[2]], unlist(nim.models[[i]],
        F)[[4]], unlist(nim.models[[i]], F)[[6]])
}

# model weights table
WAIC_nim$deltaWAIC <- WAIC_nim$WAIC - min(WAIC_nim$WAIC)
WAIC_nim$rel_like <- exp(-0.5 * WAIC_nim$deltaWAIC)
WAIC_nim$weight <- WAIC_nim$rel_like/sum(WAIC_nim$rel_like)
```

```
WAIC_nim[order(-WAIC_nim$weight), ]
```

	modname	WAIC	deltaWAIC	rel_like	weight
4	nimblepigs.SowAge	315.88	0.00	1	0.99
2	nimblepigs.sexyear	326.60	10.72	0	0.00
3	nimblepigs.Sow	330.25	14.36	0	0.00
1	nimblepigs.null	398.91	83.03	0	0.00

Once again, SowAge comes out on top!

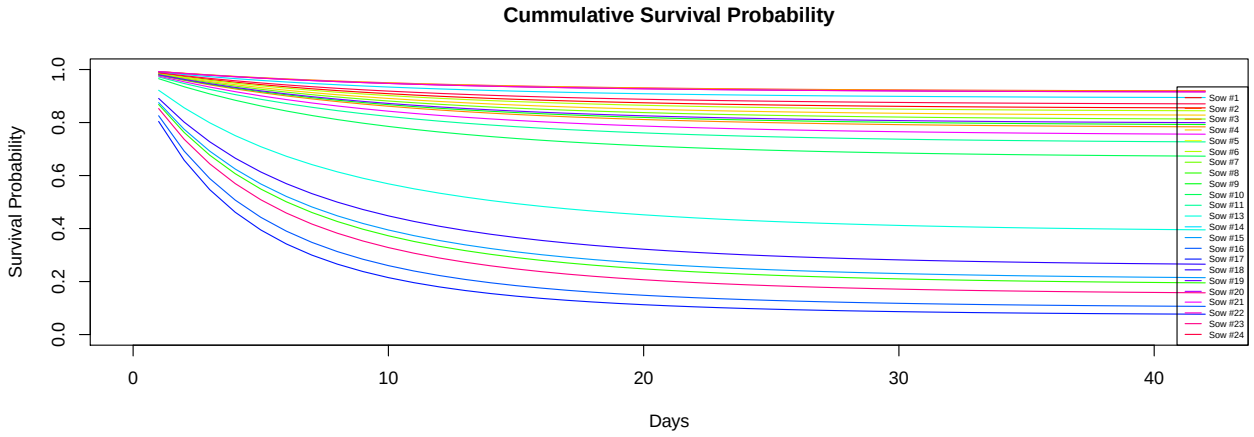
Graphing NIMBLE results

Graphing from NIMBLE isn't any different but just for kicks, here's the code to do it.

```
variables <- summary(sowage.outputs)$quantiles
daily.surv <- cum.surv <- matrix(NA, nrow = n.sow, ncol = 42)
for (i in 1:n.sow) {
    # plogis(beta.0 + beta.Sow + beta.age*age)
    daily.surv[i, ] <- plogis(variables[31, 3] + variables[i, 3] + variables[24,
        3] * 1:42)
    cum.surv[i, ] <- cumprod(daily.surv[i, ])
}
```

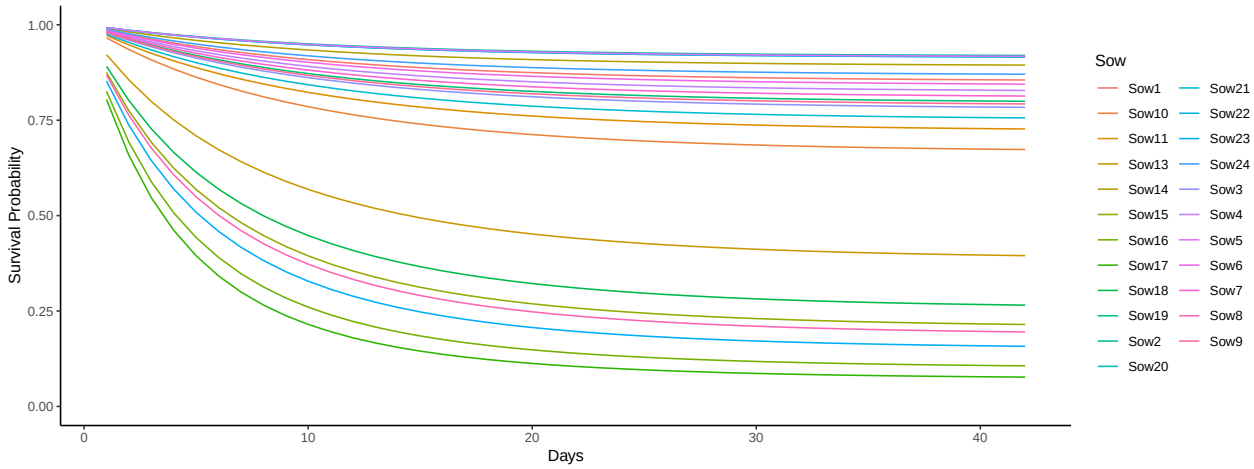
For base R:

```
plot(1:42, cum.surv[1, ], type = "l", col = rainbow(n.sow)[1], xlim = c(0, 42), ylim = c(0,
    1), main = "Cummulative Survival Probability", xlab = "Days", ylab = "Survival Probability")
for (i in 2:n.sow) {
    lines(1:42, cum.surv[i, ], type = "l", col = rainbow(n.sow)[i])
}
legend("bottomright", paste("Sow #", c(1:11, 13:24), sep = ""), lty = 1, col = rainbow(n.sow),
    cex = 0.5)
```




```
library(ggplot2)
library(tidyr)
cum.surv <- as.data.frame(cum.surv)
cum.surv$Sow = paste("Sow", c(1:11, 13:24), sep = "")
long_surv <- pivot_longer(cum.surv, cols = starts_with("V"))
long_surv$age <- rep(1:42, n.sow)

print(ggplot(long_surv, aes(x = age, y = value, col = Sow, group = Sow)) + geom_line() +
    theme_classic() + ylim(0, 1) + xlab("Days") + ylab("Survival Probability"))
```



Whooo look at that gorgeous curve!