

## A cautionary note on the use of the categorical and multinomial distributions in Nimble and JAGS

Michael Schaub, 14 Aug 2025

The categorical distribution is used to model discrete outcomes from an exhaustive set of possible outcomes, with the aim of estimating the probability of each outcome occurring. Since the outcome must be in the set, the sum of these probabilities must equal one. The same is true for the multinomial distribution, where we model the frequency of each possible outcome. I always thought that Nimble and JAGS would throw an error if these cell probabilities in a categorical or multinomial model were specified in such a way that their sum was not equal to one... but this is not the case: both the categorical and the multinomial models will also work in this instance. This can be quite dangerous, as we will see. The purpose of this memo is to make you aware that these models must be correctly specified (i.e. the probabilities must sum to 1); otherwise, biased inferences are likely to occur.

To illustrate the issue, I simulate a dataset. There are three possible outcomes (1, 2 or 3) that occur with probabilities of 0.1, 0.3 and 0.6, respectively. We simulate a dataset with 1,000 values and then use the categorical distribution to estimate the cell probabilities.

```
p <- c(0.1, 0.3, 0.6)
n <- 1000
y <- sample(c(1, 2, 3), n, replace=TRUE, prob=p)
```

Then I fit the model in Nimble. First, I use the correct model specification. We know that each value of  $p$  must lie between 0 and 1, and that the sum of the elements of the vector must equal 1. One way to impose these constraints is to define a Dirichlet prior for  $p$ .

```
library(nimble)
library(MCMCvis)

# Specify model in nimble
modell <- nimbleCode({
  for (i in 1:n){
    y[i] ~ dcat(p[1:3])
  }
  p[1:3] ~ ddirch(alpha[1:3])
})
```

We bundle the data and constants, and run the model.

```
data <- list(y=y)
const <- list(n=length(y), alpha=c(1, 1, 1))
parameters <- c('p')
inits <- list(p=rep(1/3, 3))
ni <- 5000; nt <- 1; nb <- 1000; nc <- 2

out1 <- nimbleMCMC(code=modell, constants=const, data=data,
inits=inits, monitors=parameters, thin=nt, niter=ni, nburnin=nb,
nchains=nc, samplesAsCodaMCMC=TRUE)

MCMCsummary(out1, digits=3)
```

	mean	sd	2.5%	50%	97.5%	Rhat	n.eff
p[1]	0.103	0.00963	0.0846	0.102	0.122	1	8000
p[2]	0.331	0.01500	0.3020	0.331	0.362	1	7601
p[3]	0.566	0.01590	0.5350	0.566	0.597	1	7770

This looks good; we successfully recover the values of the parameters.

Now, let's assume that we ignore the requirement that the probabilities must sum to 1 and simply impose the constraint that they lie between 0 and 1. Consequently, we would specify a uniform prior for each of them.

Note that in this extremely simple model, it may appear strange that such an oversight might happen, but in much more complex demographic models (such as multistate capture-recapture models), we may much more easily make an error such that the cell probabilities no longer add up to 1.

Let's do that and see what we get:

```
# Specify model in nimble
model2 <- nimbleCode({
  for (i in 1:n){
    y[i] ~ dcat(p[1:3])
  }
  for (i in 1:3){
    p[i] ~ dunif(0, 1)
  }
})

out2 <- nimbleMCMC(code=model2, constants=const, data=data,
  inits=inits, monitors=parameters, thin=nt, niter=ni, nburnin=nb,
  nchains=nc, samplesAsCodaMCMC=TRUE)

MCMCsummary(out2, digits=3)
```

	mean	sd	2.5%	50%	97.5%	Rhat	n.eff
p[1]	0.135	0.0429	0.0343	0.143	0.203	1.10	47
p[2]	0.435	0.1330	0.1150	0.464	0.623	1.12	41
p[3]	0.738	0.2190	0.1940	0.795	0.995	1.13	32

We can see that these estimates do not correspond to the parameters used to generate the data. Also, their sum exceeds one, so how is it possible that this model runs at all when the categorical is not defined for probabilities that do not sum to one? The solution is that, before the values are used in the distribution by Nimble or JAGS, they are scaled so that they do sum to one. Internally, the categorical uses:

$$p^*[1] = p[1] / (p[1] + p[2] + p[3])$$

$$p^*[2] = p[2] / (p[1] + p[2] + p[3])$$

$$p^*[3] = p[3] / (p[1] + p[2] + p[3])$$

That is, internally Nimble and JAGS use  $y \sim \text{dcat}(p^*)$ . When we perform the same calculation using the estimated values of  $p$ , we will obtain the same posterior means and standard deviation as with the correct model:

```
apply(MCMCchains(out2)/rowSums(MCMCchains(out2)), 2, mean)
```

```
      p[1]      p[2]      p[3]  
0.1028852 0.3324565 0.5646583
```

```
apply(MCMCchains(out2)/rowSums(MCMCchains(out2)), 2, sd)
```

```
      p[1]      p[2]      p[3]  
0.009761044 0.014661482 0.015527642
```

Here is another example inspired by multistate models. Let's assume that an individual has a 0.6 probability of survival. If it dies, this may be due to hunting (probability 0.3) or due to a natural cause. Therefore, the probabilities are as follows: alive (0.6); dead due to hunting  $((1-0.6)*0.3=0.12)$ ; and dead due to natural causes  $((1-0.6)*(1-0.3)=0.28)$ . As we can see, the probabilities sum to 1 as they should:  $0.6 + 0.12 + 0.28 = 1$ . Now, let's simulate such a dataset.

```
s <- 0.6  
q <- 0.3  
p <- c(s, (1-s)*q, (1-s)*(1-q))  
n <- 1000  
y <- sample(c(1, 2, 3), n, replace=TRUE, prob=p)
```

The goal here is to estimate  $s$  and  $q$ . We specify the model, using a uniform prior between 0 and 1 for each of the two parameters ( $s$  and  $q$ ) (this is the correct model).

```
# Specify model in nimble  
model3 <- nimbleCode({  
  for (i in 1:n){  
    y[i] ~ dcat(p[1:3])  
  }  
  p[1] <- s  
  p[2] <- (1-s)*q  
  p[3] <- (1-s)*(1-q)  
  s ~ dunif(0, 1)  
  q ~ dunif(0, 1)  
})
```

We bundle the data and constants, and run the model.

```
data <- list(y=y)  
const <- list(n=length(y))  
parameters <- c('s', 'q')  
inits <- function() list(s=runif(1), q=runif(1))  
ni <- 5000; nt <- 1; nb <- 1000; nc <- 2  
  
out3 <- nimbleMCMC(code=model3, constants=const, data=data,  
  inits=inits, monitors=parameters, thin=nt, niter=ni, nburnin=nb,  
  nchains=nc, samplesAsCodaMCMC=TRUE)  
  
MCMCsummary(out3, digits=3)
```

	mean	sd	2.5%	50%	97.5%	Rhat	n.eff
q	0.292	0.0230	0.249	0.292	0.339	1	1909
s	0.618	0.0152	0.588	0.618	0.647	1	1965

This looks good, as expected, since we're using the correct model. Now, let's assume that we specified the model incorrectly. Instead of using the correct definitions of vector  $p$ , we now have the following (note that  $p[3]$  is incorrect):

```
p[1] <- s
p[2] <- (1-s)*q
p[3] <- (1-s)*s*(1-q)
```

Admittedly, no one would do this, but it is not impossible to write a more complex model incorrectly. So let's see what we get then:

```
# Specify model in nimble
model4 <- nimbleCode({
  for (i in 1:n){
    y[i] ~ dcat(p[1:3])
  }
  p[1] <- s
  p[2] <- (1-s)*q
  p[3] <- (1-s)*s*(1-q)
  s ~ dunif(0, 1)
  q ~ dunif(0, 1)
})

out4 <- nimbleMCMC(code=model4, constants=const, data=data,
  inits=inits, monitors=parameters, thin=nt, niter=ni, nburnin=nb,
  nchains=nc, samplesAsCodaMCMC=TRUE)

MCMCsummary(out4, digits=3)
```

	mean	sd	2.5%	50%	97.5%	Rhat	n.eff
q	0.163	0.0191	0.127	0.163	0.200	1	829
s	0.474	0.0302	0.413	0.475	0.532	1	836

Now we have completely wrong estimates of  $s$  and  $q$ .

The same behaviour demonstrated for the categorical distribution also applies when the multinomial is used. I also checked the issue in JAGS, and the result was the same.

## Conclusions:

- The categorical and multinomial distributions implemented in Nimble and JAGS are highly flexible and can handle probability vectors that do not sum to one, with some elements potentially being even negative. While this may be advantageous in certain situations, it can be dangerous if we are unaware of it.
- To avoid biased estimates, ensure that the probabilities in categorical or multinomial distributions sum to 1. Depending on the context, this can be achieved using appropriate priors (e.g. the Dirichlet prior), link functions (e.g. the multinomial link function, which is not shown here), or by specifying the probabilities correctly as functions of other parameters, as is often the case in multistate models.