

Fitting N-mixture models with random observer effects in BUGS and ADMB

Richard Chandler, Marc Kéry & Hans Skaug, 20 October 2012

1. Introduction

Abundance is the central state variable in much of ecology and its applications, such as wildlife management and conservation biology. Typically, abundance must be estimated owing to imperfect detection of the animals in the wild: some individuals will be overlooked and thus, counts of animals are rarely equal to true abundance. Instead, abundance and counts are linked by detection probability, which in this context is the probability that an individual present appears in a count. A vast array of sampling designs and models have been developed over the last 50 years to estimate abundance, e.g. Buckland et al. (2001), Borchers et al. (2002), Williams et al. (2002) and Royle and Dorazio (2008). Most protocols to estimate abundance require that individuals can be individually identified, a condition which often requires capturing and marking of animals. This is costly and therefore the N-mixture, or binomial mixture, model of Royle (2004) is an appealing alternative: this model yields estimates of abundance from spatially and temporally replicated counts of *unmarked* animals alone. Typical applications of the Nmixture model require the assumption of some effects as random, for instance, to account for intrinsic differences in the ability of field ornithologists to detect and identify birds.

In section 2, this report provides a brief description of the Nmixture model and in section 3 R code is shown to simulate data with a fixed covariate effect and a random observer effect on detection probability. In section 4, the Nmixture model is fitted to one simulated data set using WinBUGS (or JAGS) called from R and in section 5, the same is done for [AD Model Builder with its random effects module, referred to here](#) ADMB-RE. Section 6 compares the results from the two programs and Section 7 briefly discusses the main challenges encountered when translating BUGS code to ADMB.

2. Model description

We assume our data y_{it} are the number y of individuals counted at site i ($i = 1 \dots R$) at time t ($t = 1 \dots T$). We make the assumption that the total period over which the T replicated counts are made is sufficiently short that the population size N_i at each site may be assumed unchanged. The relationship between the observed counts y_{it} and the underlying abundances at each site i , N_i , may be described by just two equations. The first equation, the state equation, describes the latent abundance N_i at site i and the second, the observation equation, describes the observed counts y_{it} as a function of that latent abundance:

1. State equation: $N_i \sim \text{Poisson}(\lambda_i)$
2. Observation equation: $y_{it} | N_i \sim \text{Binomial}(N_i, p_{it})$

In the state equation, λ_i is the expected abundance at site i and the Poisson distribution is the standard description of the spatial variation of abundance. Possible alternatives for a Poisson include the negative binomial (Royle 2004), Poisson-lognormal (Royle and Dorazio 2008) or a zero-inflated Poisson distribution (Wenger and Freeman 2008; Kéry and Schaub 2011). The expected abundance λ_i is indexed i , allowing it to vary by site. Thus, unstructured heterogeneity among sites could be modelled by incorporating random site effects, resulting in a Poisson-lognormal distribution in the state equation. Alternatively, the effects of recognized and measured covariates that vary among sites ('site covariates') could be modeled linearly in a GLM fashion via a log link function.

In the observation equation, we assume a Binomial distribution with trial size N_i and success probability p_{it} to describe the variability of the observed counts y_{it} . The parameter p_{it} represents the detection probability. The choice of a Binomial observation process implies the assumption that there are no false-positive errors, i.e. no double counts or other species counted erroneously as the target species. Since p_{it} is indexed by both site i and survey t , we can let it vary by site or survey or by a combination of site and survey by modeling unstructured heterogeneity via random site, survey or site-by-survey effects. Similarly, effects of site covariates or of covariates varying by site and survey ('survey covariates') can be modeled in a linear GLM fashion via a logit link. Of course, in some cases we may simply assume that either or both parameters of the mixture model are constant, i.e. $\lambda_i = \lambda$ and $p_{it} = p$.

In this report, we adopt a state equation with constant λ and an observation equation with a fixed effect p_1 of a site covariate x and an observer random effect u_k .

$$N_i \sim \text{Poisson}(\lambda)$$

$$y_{it} | N_i \sim \text{Binomial}(N_i, p_{it})$$

$$\text{logit}(p_{it}) = p_0 + p_1 * x_i + u_{k(it)}, \text{ with } u_k \sim \text{Normal}(0, \sigma)$$

$u_{k(it)}$ is the effect of observer k surveying site i at time t . We assume that u_k is a draw from a zero-mean normal distribution with standard deviation σ . The parameters to be estimated are: $\log(\lambda)$, p_0 , p_1 , $\log(\sigma)$ and the random observer effects u_k .

3. Simulating data and example data set

Here is R code to simulate data sets for R sites that are surveyed T times and with specified values of the parameters λ , p_0 , p_1 and σ .

```
data.fn <- function(R = 100, T = 10, lambda = 5, p0 = 1, p1 = 1, pSD
= 0.5, nG = 40, xSD = 1){
#   R: number of surveyed sites
#   T: number of surveys at each site
#   lambda: expected abundance
#   p0: intercept of detection probability (logit scale)
#   p1: slope of detection probability on covariate x (logit scale)
#   pSD: standard deviation of normal distribution from
#   which observer effects u are drawn
#   nG: number of different observers (called K in section 2)
#   xSD: standard deviation of zero-mean normal distribution from
#   which covariate values x are generated

# State equation: Generate abundance from Poisson distribution
N <- rpois(R, lambda)

# Draw values of covariate X
x <- rnorm(R, 0, xSD)

# Generate observer ID array
gID <- matrix(sort(rep(1:nG, 25)), nrow = R, ncol = T, byrow = TRUE)

# Draw values of covariate X
x <- rnorm(R, 0, xSD)

# Draw values of observer effects u and put them in R-T matrix
u0 <- rnorm(nG, 0, pSD)
u <- matrix(rep(u0, each = 25), nrow = R, byrow = TRUE)

# Compute R-T matrix of detection probability
p <- plogis(p0 + p1 * x + u)

# Observation equation: Generate counts from Binomial distribution
y <- array(dim = c(R, T))
for(t in 1:T){
  y[,t] <- rbinom(R, N, p)
}

# Return stuff
return(list(R=R, T=T, lambda=lambda, N=N, nG=nG, p0=p0, p1=p1, p=p,
gID=gID, pSD=pSD, x=x, y=y, xSD=xSD, u=u, u0=u0))
}
```

To generate one data set, we execute the function definition in R and then call the function like this:

```
simdata <- data.fn()
str(simdata)
> str(simdata)
List of 15
```

```

$ R      : num 100
$ T      : num 10
$ lambda: num 5
$ N      : num [1:100] 3 9 5 5 5 8 6 6 3 7 ...
$ nG     : num 40
$ p0     : num 1
$ p1     : num 1
$ p      : num [1:100, 1:10] 0.442 0.754 0.919 0.92 0.888 ...
$ gID    : int [1:100, 1:10] 1 1 1 2 2 3 3 3 4 4 ...
$ pSD    : num 0.5
$ x      : num [1:100] -1.749 -0.4 0.916 1.188 0.819 ...
$ y      : num [1:100, 1:10] 2 8 5 5 5 7 4 2 3 5 ...
$ xSD    : num 1
$ u      : num [1:100, 1:10] 0.518 0.518 0.518 0.256 0.256 ...
$ u0     : num [1:40] 0.5175 0.2559 -0.0373 -1.0147 0.4959 ...

```

In this report, we use the data set contained in the R script called `simNmix.R`. We load it into an R workspace:

```
source("G:\\Nmix-ADMB documentation with Richard\\simNmix.R")
```

The R workspace now contains the following objects:

```

ls()
> ls()
 [1] "gID"      "lambda"  "N"       "nG"      "p"       "p0"      "p1"
"pSD"      "R"       "T"       "u"       "u0"      "x"       "y"

```

Now we are ready to fit the model.

4. Code for analysis using BUGS or JAGS when called from R and solutions for example data set

We use WinBUGS (or JAGS) called from R, using functionality in the R2WinBUGS package (Sturtz et al. 2005) to fit the model to the data set just loaded. We first write into the R working directory a text file containing the BUGS code to fit the model. Later we define other R objects that contain the data, initial values, a list of the parameters to be estimated and the MCMC settings. Functions `bugs()` (or `jags()`) take all of these and sends them to WinBUGS. After execution of the Markov chain Monte Carlo (MCMC) analysis in WinBUGS, results are imported back into the R workspace and can be inspected and summarized.

```

# Define model
sink("Nmix.txt")
cat("
model {

# Priors
log(lambda) <- loglam
loglam ~ dunif(-10, 10)
p0 ~ dunif(-10, 10)
p1 ~ dunif(-10, 10)

```

```

for(k in 1:nG){
  u[k] ~ dnorm(0, tau)
}
tau <- pow(sigma, -2)
sigma ~ dunif(0, 5)

# State equation
for (i in 1:R) {
  N[i] ~ dpois(lambda)

  # Observation equation
  for (t in 1:T) {
    y[i,t] ~ dbin(p[i,t], N[i])
    p[i,t] <- 1 / (1 + exp( -1 * (p0 + p1 * x[i] + u[gID[i,t]])))
  }
}

# Derived quantities
totalN <- sum(N[]) # Population size over all R sites
logsigma <- log(sigma)
}
",fill=TRUE)
sink()

# Bundle data
attach(simdata)
win.data <- list(y = y, R = R, T = T, x = x, gID = gID, nG = nG)

# Inits function
Nst <- apply(y, 1, max) + 1
inits <- function(){list(N = Nst, sigma = rlnorm(1))}

# Parameters to be estimated
params <- c("lambda", "loglam", "p0", "p1", "sigma", "logsigma",
"u", "totalN")
# Could include "N" to get local population size

# MCMC settings (takes about 13 min on a slow laptop)
nc <- 3
nb <- 5000
ni <- 10000
nt <- 5

# MCMC test settings
# nc <- 3 ; nb <- 20 ; ni <- 120 ; nt <- 5

# Call WinBUGS from R (bugs run time = 7.88 on my desktop)
out <- bugs(win.data, inits, params, "Nmix.txt", n.chains=nc,
n.iter=ni, n.burn = nb, n.thin=nt, debug = TRUE, bugs.directory =
"c:/Programme/WinBUGS14/")
brt()

```

Summarize marginal posteriors

```
print(out, 3)
```

```
> print(out, 3)
```

```
Inference for Bugs model at "Nmix.txt", fit using WinBUGS,
```

```
3 chains, each with 10000 iterations (first 5000 discarded), n.thin = 5
```

```
n.sims = 3000 iterations saved
```

	mean	sd	2.5%	25%	50%	75%	97.5%	Rhat	n.eff
lambda	4.813	0.242	4.340	4.644	4.814	4.972	5.293	1.003	710
loglam	1.570	0.050	1.468	1.536	1.571	1.604	1.666	1.003	730
p0	0.736	0.093	0.551	0.674	0.736	0.795	0.927	1.004	1300
p1	1.029	0.072	0.893	0.979	1.028	1.077	1.174	1.002	1500
sigma	0.459	0.086	0.301	0.401	0.455	0.513	0.639	1.003	680
logsigma	-0.796	0.191	-1.199	-0.913	-0.787	-0.667	-0.447	1.003	680
u[1]	-0.361	0.254	-0.862	-0.531	-0.361	-0.188	0.130	1.001	3000
u[2]	-0.087	0.271	-0.616	-0.272	-0.085	0.112	0.423	1.001	3000
u[3]	-0.147	0.267	-0.692	-0.322	-0.139	0.035	0.367	1.005	510
u[4]	0.555	0.297	-0.042	0.359	0.550	0.755	1.125	1.003	870
u[5]	0.200	0.241	-0.270	0.039	0.204	0.360	0.671	1.004	540
u[6]	0.268	0.257	-0.220	0.085	0.267	0.445	0.765	1.002	1700
u[7]	-0.157	0.227	-0.622	-0.304	-0.155	-0.009	0.297	1.001	3000
u[8]	0.412	0.294	-0.163	0.214	0.412	0.613	0.974	1.004	660
u[9]	0.188	0.235	-0.281	0.030	0.193	0.345	0.649	1.001	3000
u[10]	0.510	0.334	-0.116	0.270	0.500	0.739	1.172	1.002	3000
u[11]	-0.754	0.298	-1.367	-0.951	-0.737	-0.544	-0.205	1.005	460
u[12]	0.505	0.323	-0.099	0.276	0.493	0.720	1.168	1.001	2600
u[13]	-0.703	0.270	-1.268	-0.876	-0.688	-0.518	-0.214	1.002	1600
u[14]	-0.944	0.361	-1.681	-1.178	-0.928	-0.688	-0.281	1.001	2600
u[15]	-0.015	0.280	-0.552	-0.210	-0.018	0.175	0.560	1.001	2100
u[16]	-0.137	0.256	-0.663	-0.302	-0.127	0.035	0.325	1.002	1600
u[17]	0.024	0.357	-0.699	-0.206	0.032	0.257	0.723	1.002	1400
u[18]	0.132	0.348	-0.566	-0.090	0.136	0.369	0.795	1.001	3000
u[19]	-0.017	0.241	-0.496	-0.176	-0.023	0.150	0.446	1.001	3000
u[20]	-0.268	0.223	-0.715	-0.416	-0.263	-0.120	0.172	1.001	3000
u[21]	0.069	0.290	-0.493	-0.121	0.069	0.259	0.654	1.001	3000
u[22]	-0.321	0.256	-0.839	-0.485	-0.314	-0.153	0.162	1.001	3000
u[23]	0.149	0.279	-0.399	-0.036	0.141	0.336	0.707	1.003	720
u[24]	0.086	0.268	-0.453	-0.090	0.087	0.270	0.613	1.001	3000
u[25]	-0.243	0.282	-0.827	-0.433	-0.234	-0.049	0.293	1.005	610
u[26]	0.282	0.295	-0.328	0.099	0.286	0.480	0.862	1.001	3000
u[27]	-0.127	0.289	-0.691	-0.315	-0.128	0.061	0.450	1.001	3000
u[28]	-0.317	0.280	-0.864	-0.499	-0.323	-0.137	0.258	1.001	3000
u[29]	-0.229	0.230	-0.712	-0.372	-0.219	-0.076	0.208	1.002	1500
u[30]	-0.280	0.254	-0.798	-0.443	-0.276	-0.112	0.207	1.001	2800
u[31]	0.372	0.262	-0.164	0.198	0.380	0.548	0.850	1.001	3000
u[32]	0.137	0.280	-0.439	-0.047	0.143	0.328	0.663	1.007	330
u[33]	0.095	0.262	-0.412	-0.088	0.087	0.273	0.633	1.004	650
u[34]	0.386	0.270	-0.162	0.219	0.388	0.572	0.902	1.001	3000
u[35]	-0.131	0.291	-0.687	-0.326	-0.138	0.060	0.434	1.003	780
u[36]	0.576	0.286	0.042	0.374	0.569	0.761	1.162	1.001	3000
u[37]	-0.169	0.231	-0.624	-0.327	-0.164	-0.012	0.276	1.002	1300
u[38]	0.640	0.272	0.140	0.450	0.633	0.826	1.193	1.001	3000
u[39]	-0.079	0.230	-0.552	-0.229	-0.077	0.075	0.381	1.001	3000
u[40]	-0.167	0.238	-0.638	-0.323	-0.161	-0.012	0.284	1.001	3000
totalN	481.793	10.241	463.975	475.000	481.500	488.000	503.000	1.006	340
deviance	2466.096	14.886	2439.000	2456.000	2465.000	2476.000	2498.000	1.001	3000

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, $pD = \text{var}(\text{deviance})/2$)

$pD = 110.8$ and $DIC = 2576.9$

DIC is an estimate of expected predictive error (lower deviance is better).

Call JAGS from R (run time = 7.9 min on same desktop)

have to installed JAGS software and installed

R packages **rjags** and **R2jags**.

library(R2jags)

```

system.time(
  outJ <- jags(win.data, inits, params, "Nmix.txt", n.chains=nc,
    n.iter=ni, n.burn = nb, n.thin=nt)
)
traceplot(outJ)          # Look at convergence

```

Summarize marginal posteriors

```
print(outJ, 3)
```

```

Inference for Bugs model at "Nmix.txt", fit using jags,
  3 chains, each with 10000 iterations (first 5000 discarded), n.thin = 5
  n.sims = 3000 iterations saved

```

	mu.vect	sd.vect	2.5%	25%	50%	75%	97.5%	Rhat	n.eff
lambda	4.813	0.242	4.354	4.656	4.808	4.969	5.313	1.001	3000
loglam	1.570	0.050	1.471	1.538	1.570	1.603	1.670	1.001	3000
logsigma	-0.797	0.185	-1.181	-0.916	-0.792	-0.672	-0.448	1.001	3000
p0	0.732	0.091	0.550	0.673	0.733	0.795	0.909	1.001	2700
p1	1.028	0.072	0.888	0.979	1.029	1.075	1.171	1.002	1900
sigma	0.458	0.085	0.307	0.400	0.453	0.511	0.639	1.001	3000
totalN	481.247	10.242	463.000	474.000	481.000	488.000	503.000	1.001	3000
u[1]	-0.355	0.256	-0.872	-0.531	-0.356	-0.181	0.130	1.001	3000
u[2]	-0.095	0.269	-0.618	-0.277	-0.092	0.090	0.430	1.002	1300
u[3]	-0.158	0.256	-0.695	-0.327	-0.160	0.010	0.332	1.001	2500
u[4]	0.555	0.297	0.005	0.349	0.553	0.762	1.138	1.003	680
u[5]	0.199	0.246	-0.290	0.039	0.191	0.364	0.689	1.001	2800
u[6]	0.290	0.262	-0.211	0.113	0.283	0.464	0.815	1.001	3000
u[7]	-0.166	0.231	-0.617	-0.321	-0.161	-0.011	0.302	1.001	3000
u[8]	0.410	0.298	-0.165	0.195	0.410	0.622	0.971	1.002	1600
u[9]	0.204	0.236	-0.283	0.050	0.207	0.368	0.666	1.001	2700
u[10]	0.525	0.330	-0.099	0.292	0.516	0.752	1.186	1.001	3000
u[11]	-0.763	0.295	-1.394	-0.952	-0.763	-0.557	-0.226	1.001	2700
u[12]	0.508	0.313	-0.085	0.294	0.493	0.717	1.131	1.001	3000
u[13]	-0.704	0.262	-1.258	-0.874	-0.689	-0.525	-0.230	1.004	550
u[14]	-0.939	0.356	-1.685	-1.177	-0.926	-0.685	-0.277	1.001	3000
u[15]	0.002	0.282	-0.550	-0.182	0.004	0.192	0.554	1.002	1000
u[16]	-0.117	0.248	-0.630	-0.276	-0.106	0.052	0.345	1.002	1600
u[17]	0.048	0.353	-0.662	-0.183	0.055	0.289	0.713	1.001	3000
u[18]	0.144	0.345	-0.556	-0.082	0.146	0.370	0.826	1.002	1200
u[19]	-0.013	0.238	-0.501	-0.165	-0.018	0.146	0.443	1.002	1600
u[20]	-0.262	0.224	-0.716	-0.406	-0.262	-0.110	0.157	1.001	3000
u[21]	0.080	0.304	-0.505	-0.127	0.079	0.275	0.681	1.001	3000
u[22]	-0.309	0.253	-0.812	-0.474	-0.303	-0.137	0.166	1.002	1900
u[23]	0.154	0.278	-0.386	-0.028	0.153	0.338	0.721	1.003	2500
u[24]	0.094	0.268	-0.450	-0.085	0.099	0.275	0.614	1.001	3000
u[25]	-0.239	0.288	-0.824	-0.420	-0.228	-0.047	0.307	1.001	3000
u[26]	0.263	0.301	-0.375	0.074	0.281	0.467	0.809	1.001	3000
u[27]	-0.120	0.289	-0.675	-0.304	-0.118	0.064	0.483	1.001	3000
u[28]	-0.307	0.286	-0.859	-0.498	-0.302	-0.118	0.242	1.001	3000
u[29]	-0.222	0.221	-0.682	-0.364	-0.218	-0.074	0.197	1.002	1400
u[30]	-0.261	0.251	-0.766	-0.425	-0.253	-0.084	0.221	1.001	3000
u[31]	0.369	0.263	-0.167	0.198	0.368	0.544	0.874	1.002	1800
u[32]	0.139	0.279	-0.414	-0.047	0.146	0.331	0.672	1.001	3000
u[33]	0.102	0.259	-0.397	-0.071	0.096	0.273	0.624	1.001	3000
u[34]	0.398	0.264	-0.127	0.225	0.396	0.576	0.909	1.001	3000
u[35]	-0.127	0.286	-0.680	-0.318	-0.134	0.059	0.448	1.001	3000
u[36]	0.573	0.284	0.050	0.379	0.564	0.756	1.166	1.002	1100
u[37]	-0.168	0.225	-0.612	-0.319	-0.165	-0.017	0.266	1.001	3000
u[38]	0.642	0.269	0.112	0.454	0.637	0.819	1.184	1.001	3000
u[39]	-0.082	0.226	-0.546	-0.234	-0.077	0.069	0.352	1.001	3000
u[40]	-0.161	0.241	-0.640	-0.321	-0.159	0.007	0.294	1.001	2200
deviance	2466.110	15.042	2438.385	2455.543	2465.565	2475.618	2498.301	1.001	3000

For each parameter, n.eff is a crude measure of effective sample size, and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

```
DIC info (using the rule, pD = var(deviance)/2)
```

```
pD = 113.2 and DIC = 2579.3
```

DIC is an estimate of expected predictive error (lower deviance is better).

5. Code for analysis using ADMB-RE

A primary difference between ADMB and BUGS is that ADMB does not allow discrete latent random variables (N's). Hence, the users must write out the likelihood of the model (in C++). This is a good thing because it forces the analyst to get a clear understanding of the likelihood of the model, but is much more difficult than fitting the model in BUGS, which requires only a symbolic description of the model. The likelihood of the N-mixture model without random effects is

$$L(\lambda, p | \{Y_{ij}\}) = \prod_i^M \left\{ \sum_{N_i=\max(Y_i)}^{\infty} \left(\prod_j^J \frac{N_i!}{(N_i - Y_{ij})!} p^{Y_{ij}} (1-p)^{N_i - Y_{ij}} \right) \frac{e^{-\lambda} \lambda^{N_i}}{N_i!} \right\} \quad (\text{Eq 1})$$

Notice that this is the marginal likelihood obtained by summing out the discrete random effects N_i . In practice, the upper bound of the sum is set to some number (K) higher than the maximum possible value of N, which can be found by ensuring that the MLEs are unchanged at higher values of K.

To code this model in ADMB, one must translate Eq 1 into C++. It is not helpful to try to translate the BUGS model into ADMB because the BUGS model does not show the calculations needed to evaluate the integrated likelihood. For example, in the BUGS code, there is no evidence of the product-binomial formulation or of the summation over the possible N_i because these steps are not needed to implement Gibbs sampling.

In ADMB, two files are required: the .tpl and .dat files, which are attached and annotated. Below is the PROCEDURE_SECTION of the .tpl file in which the likelihood shown in Eq 1 is specified with the additional random observer effects. Again, note that this looks nothing at all like the BUGS model file. If computer memory limitations were not a problem, the code would be much simpler because the SEPARABLE_FUNCTIONS (SF) would not be needed. However, ADMB is memory intensive especially when random effects are being estimated, and the SFs allow the problem to be partitioned into smaller pieces. The first SF adds the likelihood contribution of the random effects, which have been scaled to $u_o \sim \text{Normal}(0, 1)$; this facilitates model convergence. This transformation still allows $p_o \sim \text{Normal}(\mu, \sigma^2)$ because σ is multiplied by u_o in the linear model. See the ADMB-RE Manual for more information about this useful transformation.

The second SF evaluates the likelihood conditional on each random observer effect. Specifically, f(k) is the Poisson density (with mean lambda) for each possible value of N. g(k) is the binomial density of the observations for each value of N and probabilities p(j) which are a function of the covariate x and the fixed and random effects. The likelihood is given

by the sum of the product of f and g. ~~What more should be said here?~~
~~People can either read C++ or they can't.~~

PROCEDURE_SECTION

```
for(int i=1;i<=nG;i++)
  prior_N01(u(i));
```

```
for(int i=1;i<=R;i++)
  nll_group(i, p0,p1,log_lambda,log_sigma,u(ID(i)));
```

```
SEPARABLE_FUNCTION void prior_N01(const dvariable& u)
  nll += 0.5*square(u);
```

```
SEPARABLE_FUNCTION void nll_group(int i, const dvariable& p0,const
dvariable& p1,const dvariable& log_lambda, const dvariable& log_sigma,
const dvar_vector& u)
```

```
dvariable sigma = exp(log_sigma);
dvariable lambda = exp(log_lambda);
```

```
dvar_vector p(1,T);
dvar_vector f(1,S);
dvar_vector g(1,S);
```

```
for(int k=1;k<=S;k++)
  f(k) = pow(lambda, N(k)) / exp(lambda + gammln_tab(N(k)+1));
```

```
for(int j=1;j<=T;j++)
  p(j) = 1.0/(1.0+exp(-1.0*(p0 + p1*x(i) + sigma*u(IDind(i,j)))));
```

```
for(int k=1;k<=S;k++) {
  g(k) = 1.0;
  for(int j=1;j<=T;j++) {
    if(N(k)>=y(i,j))
      g(k) *= exp(gammln_tab(N(k)+1) - gammln_tab(y(i,j)+1) -
                  gammln_tab(N(k)-y(i,j)+1)) * pow(p, y(i,j)) *
                  pow(1.0-p, N(k)-y(i,j));
    else
      g(k) = 0.0;
  }
}
```

```
nll -= log(1e-20 + f*g);
```

TOP_OF_MAIN_SECTION

```
armb_size = 40000000L;
gradient_structure::set_GRADSTACK_BUFFER_SIZE(30000000);
gradient_structure::set_CMPDIF_BUFFER_SIZE(20000000);
```

6. Comparison of results

The following table compares the truth in the data-generation process with the estimates from WinBUGS and ADMB-RE. For the former, posterior means and standard deviations are given. Fig. 1 provides this comparison for the 40 random observer effects.

```
Truth bugs.mean bugs.sd admb.est admb.se
```

log(lambda)	1.61	1.61	0.05	1.61	0.05
p0	1.00	1.02	0.10	1.02	0.10
p1	1.00	0.96	0.07	0.95	0.07
log(sigma)	-0.69	-0.66	0.16	-0.71	0.16
u1	-0.22	-0.26	0.25	-0.45	0.48
u2	0.12	0.37	0.31	0.74	0.66
u3	0.30	0.11	0.27	0.31	0.53
u4	-0.06	-0.37	0.25	-0.68	0.50
u5	-1.03	-0.65	0.29	-1.22	0.61
u6	0.29	0.53	0.27	1.09	0.56
u7	0.24	-0.07	0.34	0.14	0.70
u8	-0.50	-0.59	0.28	-1.17	0.57
u9	0.63	0.58	0.29	1.20	0.54
u10	0.56	0.53	0.24	1.07	0.48
u11	0.42	0.06	0.28	0.24	0.50
u12	0.81	0.22	0.31	0.61	0.55
u13	0.22	0.23	0.28	0.55	0.54
u14	-1.15	-0.63	0.25	-1.25	0.50
u15	-0.04	-0.09	0.29	-0.12	0.53
u16	-0.26	-0.11	0.28	-0.18	0.58
u17	-0.21	-0.53	0.30	-1.03	0.58
u18	0.15	0.13	0.22	0.37	0.41
u19	-0.02	-0.32	0.29	-0.62	0.55
u20	0.05	0.16	0.30	0.31	0.57
u21	1.23	0.88	0.34	1.73	0.66
u22	-0.32	-0.23	0.25	-0.40	0.50
u23	0.61	0.74	0.28	1.47	0.53
u24	0.36	0.42	0.31	0.93	0.57
u25	-0.10	0.01	0.28	-0.01	0.58
u26	0.71	0.41	0.26	0.83	0.51
u27	0.64	0.62	0.30	1.29	0.65
u28	0.29	-0.05	0.28	-0.08	0.55
u29	0.64	0.58	0.28	1.26	0.49
u30	-0.44	-0.41	0.23	-0.78	0.46
u31	-0.07	-0.06	0.28	-0.08	0.57
u32	-0.60	-0.50	0.26	-1.00	0.51
u33	0.16	0.14	0.26	0.31	0.51
u34	-0.54	-0.45	0.30	-0.89	0.60
u35	-0.26	-0.45	0.28	-0.83	0.55
u36	-0.04	0.21	0.27	0.47	0.54
u37	-0.17	0.02	0.33	-0.02	0.67
u38	-1.13	-0.89	0.26	-1.81	0.52
u39	-0.14	-0.19	0.29	-0.32	0.60
u40	0.14	0.06	0.31	0.14	0.73

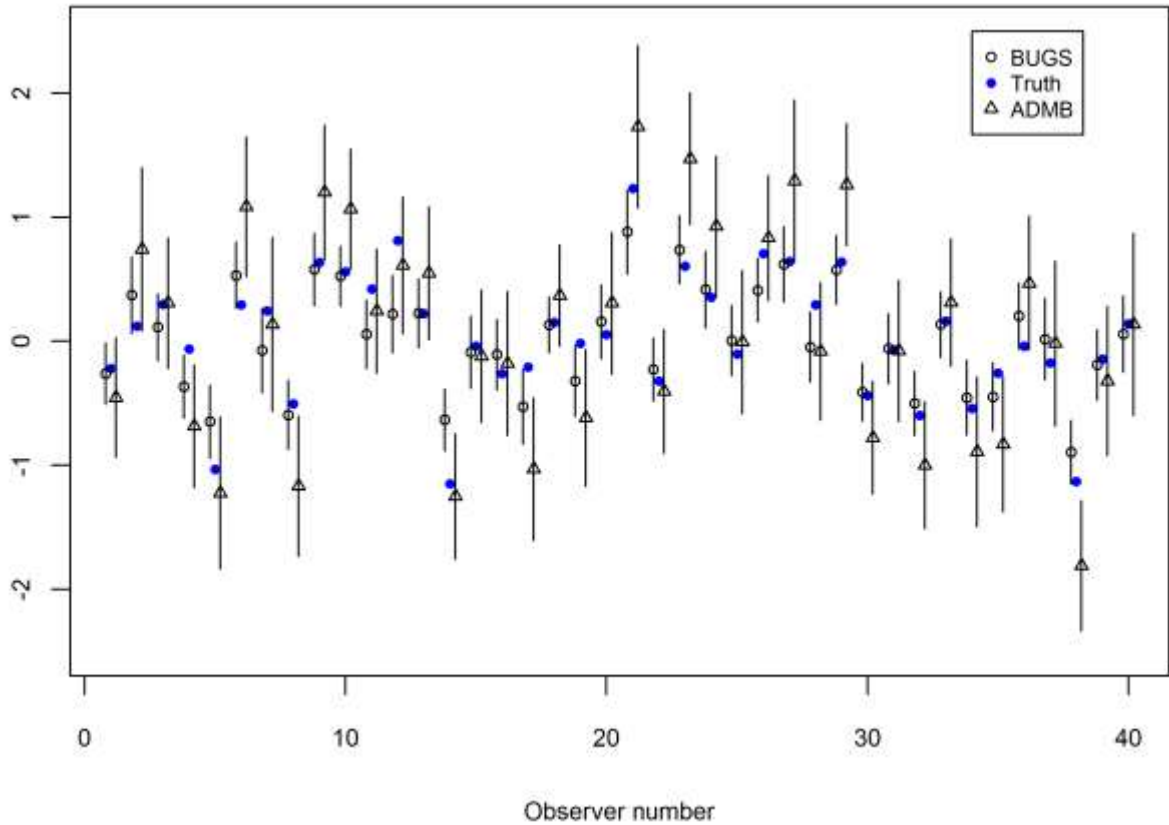


Figure 1. Comparison of the estimates (\pm SD) of the observer random effects.

A few patterns are apparent from Table 1 and Fig 1. First, the BUGS and ADMB estimates of the fixed effects and variances are very similar. Second, differences are more pronounced in the estimates of the random effects. The posterior means from WinBUGS were more accurate than the ADMB estimates (RMSE = 0.22 vs 0.43). Third, the BUGS estimates were more precise than ADMB estimates of the random effects. On average, the posterior standard deviations obtained by BUGS were 47% smaller than the standard deviations estimated by ADMB.

Not shown in Table 1 and Fig 1 are the estimates of each N_i , which can be obtained by BUGS but not ADMB. This is an important advantage of BUGS; however, we have shown that it is possible to integrate out the discrete random effects while estimating continuous random effects. Thus ADMB represents one of the only available software for fitting such complex models within the frequentist framework. (Note that ADMB allows for Bayesian inference too.) Furthermore, ADMB is much faster than WinBUGS, in our example 4 vs. 12 mins. However, in this case, WinBUGS was easier to use and provided more precise and accurate estimates of the random effects.

7. Main challenges encountered in translating BUGS code to ADMB

The main challenges are as follows:

1. One must know something about C++ and the classes used by ADMB.
2. ADMB requires that the user know how to write a likelihood function. My guess is that many WinBUGS users could not write even a simple likelihood.
3. ADMB requires much more memory than does WinBUGS.

References

- Borchers, D. L., Buckland, S. T., and Zucchini, W. 2002. *Estimating Animal Abundance*. Springer, London.
- Buckland, S. T., Anderson, D. R., Burnham, K. P., Laake, J. L., Borchers, D. L., and Thomas, L. 2001. *Introduction to distance sampling*. Oxford University Press, Oxford.
- Kéry, M., and Schaub, M. 2011. *Bayesian population analysis using WinBUGS – a hierarchical perspective*. Academic Press.
- Royle, J. A. 2004. N-mixture models for estimating population size from spatially replicated counts. *Biometrics* 60: 108–115.
- Royle, J. A., and Dorazio, R. M. 2008. *Hierarchical modeling and inference in ecology*. Academic Press, Amsterdam.
- Sturtz, S., Ligges, U., and Gelman, A. 2005. R2WinBUGS: A package for running WinBUGS from R. *Journal of Statistical Software* 12: 1–16.
- Wenger, S. J., and Freeman, M. C. 2008. Estimating species occurrence, abundance, and detection probability using zero-inflated distributions. *Ecology* 89: 2953–2959.
- Williams, B. K., Nichols, J. D., and Conroy, M. J. 2002. *Analysis and management of animal populations*. Academic Press, San Diego, CA.