Simulate MPD data

Anthony Davidson

```
require(runjags)
```

```
## Loading required package: runjags
```

```
## Warning: package 'runjags' was built under R version 3.6.3
```

```
require(mcmcplots)
```

```
## Loading required package: mcmcplots
```

```
## Warning: package 'mcmcplots' was built under R version 3.6.3
```

```
## Loading required package: coda
```

```
# sets paths for computers with other software components
# this code block produces an enviroment warning
# but not sure how to sort depenancies still
# Feb2020
# myPaths <- .libPaths("C:/Program Files/R/R-3.6.2/library")
# myPaths <- c(myPaths)
# .libPaths(myPaths)  # add new path
# .libPaths()
```

Motivation

Recently, I have been struggling with simulating data from complex hierarchical models. After several unsuccessful attempts in `R`, I remembered the good old times when I was using `WinBUGS` (more than 10 years already!) and the possibility to simulate data with it. I'm using `Jags` now, and a quick search in Google with 'simulating data with jags' led me to a complex example and a simple example.

Here, I illustrate the possibility to use Jags to simulate data with two examples that might be of interest to population ecologists: first a linear regression, second a Cormack-Jolly-Seber capture-recapture model to estimate animal survival (formulated as a state-space model).

Simulating data with `Jags` is convenient because you can use (almost) the same code for simulation and inference, and you can carry out simulation studies (bias, precision, interval coverage) in the same environment (namely `Jags`).

Linear regression example

We first load the packages we will need for this tutorial:

```
library(R2jags)
library(runjags)
library(mcmcplots)
```

Capture-recapture example

I now illustrate the use of Jags to simulate data from a Cormack-Jolly-Seber model with constant survival and recapture probabilities. I assume that the reader is familiar with this model and its formulation as a state-space model.

Let's simulate!

```
txtstring <- '
data{
# Constant survival and recapture probabilities
for (i in 1:nind){
   for (t in f[i]:(n.occasions-1)){
      phi[i,t] <- mean.phi
      p[i,t] <- mean.p
      } #t
   } #i
# Likelihood
for (i in 1:nind){
   # Define latent state and obs at first capture
   z[i,f[i]] <- 1
   mu2[i,1] <- 1 * z[i,f[i]] # detection is 1 at first capture ("conditional on first capture")
   y[i,1] ~ dbern(mu2[i,1])
   # then deal w/ subsequent occasions
   for (t in (f[i]+1):n.occasions){
      # State process
      z[i,t] ~ dbern(mu1[i,t])
      mu1[i,t] <- phi[i,t-1] * z[i,t-1]
      # Observation process
      y[i,t] ~ dbern(mu2[i,t])
      mu2[i,t] <- p[i,t-1] * z[i,t]
      } #t
   } #i
}
model{
fake <- 0
}
'
```

Let's pick some values for parameters and store them in a data list:

```
# parameter for simulations
n.occasions = 10 # nb of occasions
nind = 100 # nb of individuals
mean.phi <- 0.8 # survival
mean.p <- 0.6 # recapture
f = rep(1,nind) # date of first capture
data<-list(n.occasions = n.occasions, mean.phi = mean.phi, mean.p = mean.p, f = f, nind = nind)
```

Now run Jags:

```
out <- run.jags(txtstring, data = data,monitor=c("y"),sample=1, n.chains=1, summarise=FALSE)
```

```
## Compiling rjags model...
## Calling the simulation using the rjags method...
## Note: the model did not require adaptation
## Burning in the model for 4000 iterations...
```

```
## Running the model for 1 iterations...
## Simulation complete
## Finished running the simulation
```

Format the output:

```
Simulated <- coda::as.mcmc(out)
dim(Simulated)
```

```
## [1]    1 1000
```

```
dat = matrix(Simulated,nrow=nind)
head(dat)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    0    0    0    0    0    0    0    0     0
## [2,]    1    0    0    0    0    0    0    0    0     0
## [3,]    1    0    1    1    0    0    0    0    0     0
## [4,]    1    0    1    1    1    0    0    0    0     0
## [5,]    1    1    1    1    1    0    0    0    0     0
## [6,]    1    1    1    1    1    0    0    1    1     1
```

Here I monitored only the detections and non-detections, but it is also possible to get the simulated values for the states, i.e. whether an individual is alive or dead at each occasion. You just need to amend the call to `Jags` with `monitor=c("y","x")` and to amend the output accordingly.

Now we fit a Cormack-Jolly-Seber model to the data we've just simulated, assuming constant parameters:

```
model <-
paste("
model {
# Priors and constraints
for (i in 1:nind){
   for (t in f[i]:(n.occasions-1)){
      phi[i,t] <- mean.phi
      p[i,t] <- mean.p
      } #t
   } #i
mean.phi ~ dunif(0, 1)         # Prior for mean survival
mean.p ~ dunif(0, 1)           # Prior for mean recapture
# Likelihood
for (i in 1:nind){
   # Define latent state at first capture
   z[i,f[i]] <- 1
   for (t in (f[i]+1):n.occasions){
      # State process
      z[i,t] ~ dbern(mu1[i,t])
      mu1[i,t] <- phi[i,t-1] * z[i,t-1]
      # Observation process
      y[i,t] ~ dbern(mu2[i,t])
      mu2[i,t] <- p[i,t-1] * z[i,t]
      } #t
   } #i
}
")
writeLines(model,"cjs.jags")
```

Prepare the data:

```
# vector with occasion of marking
get.first <- function(x) min(which(x!=0))
f <- apply(dat, 1, get.first)
# data
jags.data <- list(y = dat, f = f, nind = dim(dat)[1], n.occasions = dim(dat)[2])
```

```
# Initial values
known.state.cjs <- function(ch){
   state <- ch
   for (i in 1:dim(ch)[1]){
      n1 <- min(which(ch[i,]==1))
      n2 <- max(which(ch[i,]==1))
      state[i,n1:n2] <- 1
      state[i,n1] <- NA
      }
   state[state==0] <- NA
   return(state)
   }
inits <- function(){list(mean.phi = runif(1, 0, 1), mean.p = runif(1, 0, 1), z = known.state.cjs(dat))}
```

We'd like to carry out inference about survival and recapture probabilities:

```
parameters <- c("mean.phi", "mean.p")
```

Standard MCMC settings:

```
ni <- 10000
nt <- 6
nb <- 5000
nc <- 2
```

Ready to run Jags!

```
# Call JAGS from R (BRT 1 min)
cjs <- jags(jags.data, inits, parameters, "cjs.jags", n.chains = nc, n.thin = nt, n.iter = ni, n.burnin
```

```
## module glm loaded
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 900
##    Unobserved stochastic nodes: 902
##    Total graph size: 3707
##
## Initializing model
```

Summarize posteriors and compare to the values we used to simulate the data:

```
print(cjs, digits = 3)
```

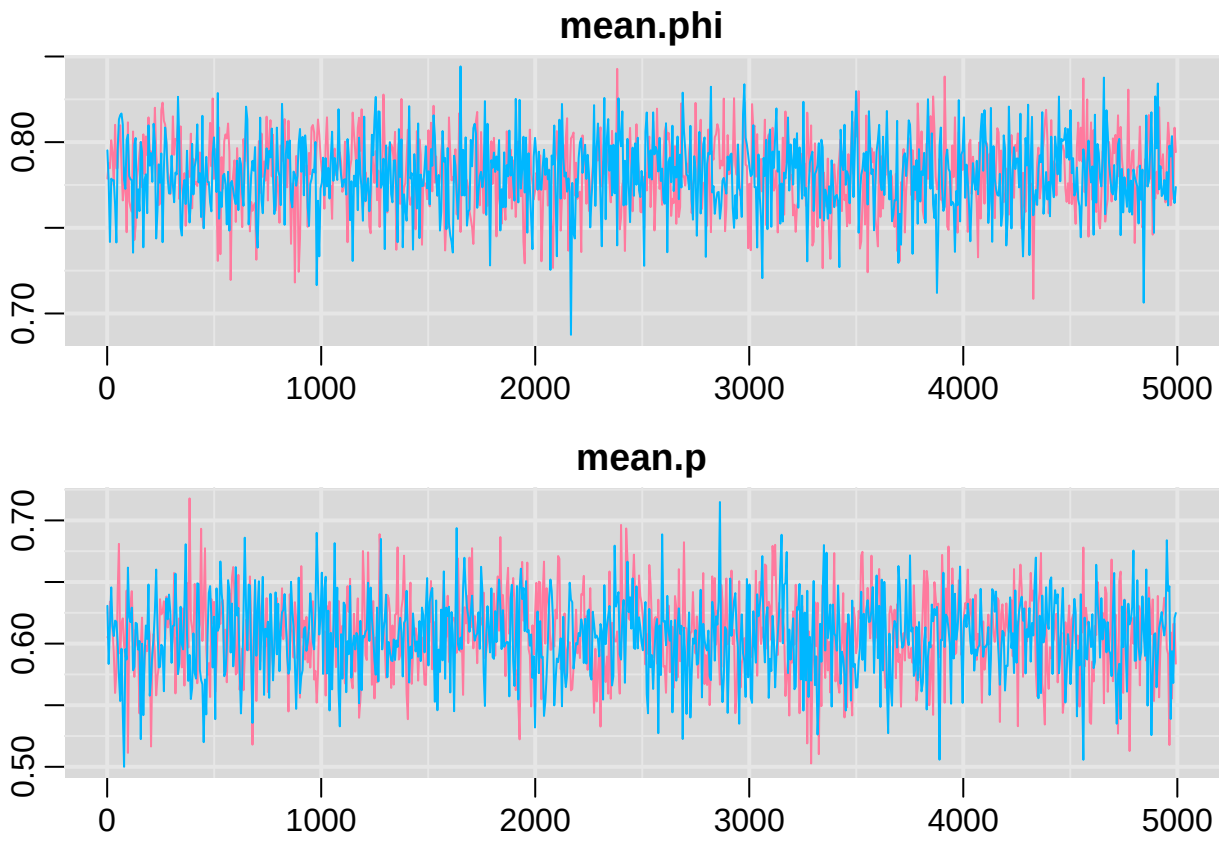
```
## Inference for Bugs model at "cjs.jags", fit using jags,
##  2 chains, each with 10000 iterations (first 5000 discarded), n.thin = 6
##  n.sims = 1666 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
## mean.p     0.607   0.032   0.542   0.587   0.608   0.628   0.671 1.002   890
## mean.phi   0.781   0.021   0.737   0.767   0.781   0.795   0.821 1.001  1700
```

```
## deviance 440.591  18.999 405.916 426.753 439.808 452.503 480.487 1.001  1700
##
## 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 = 180.6 and DIC = 621.2
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Again pretty close!

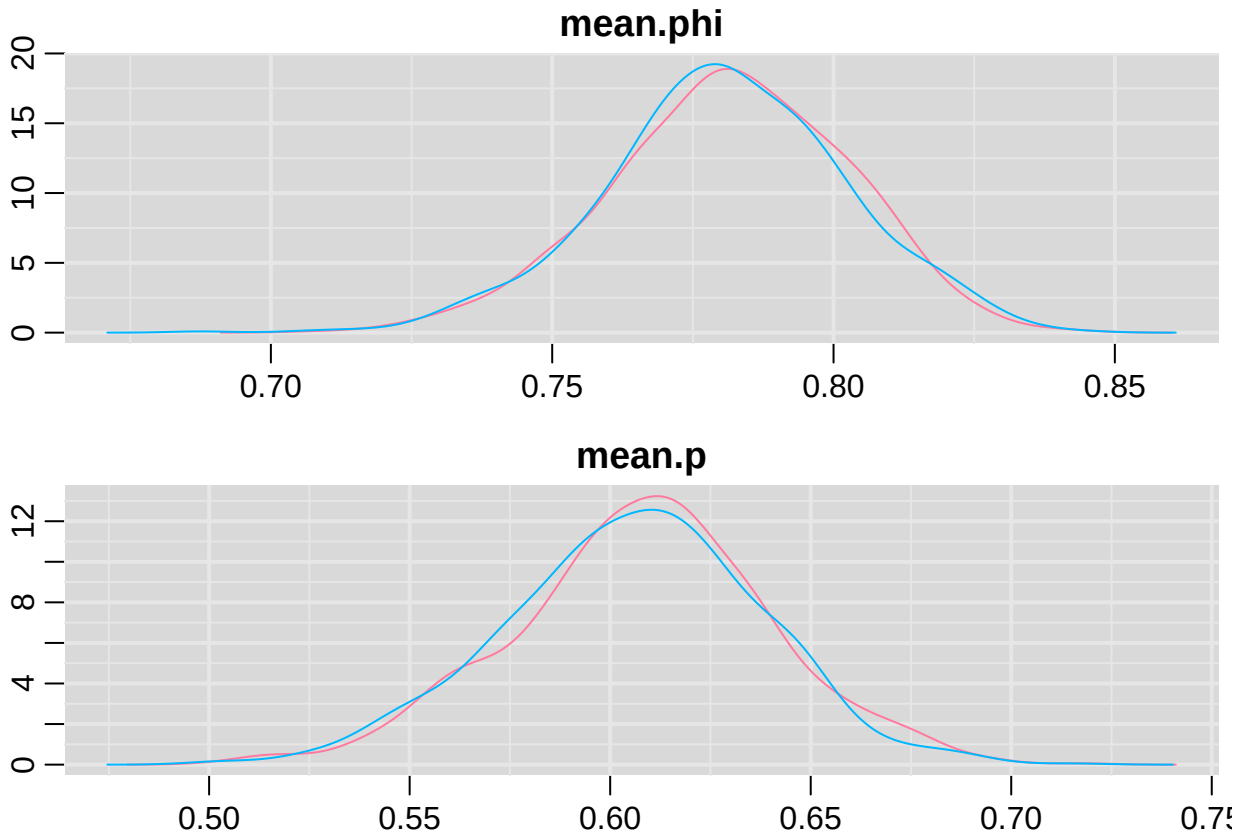
Trace plots

```
traplot(cjs,c("mean.phi", "mean.p"))
```



Posterior distribution plots:

```
denplot(cjs,c("mean.phi", "mean.p"))
```


mean.phi
0
5
10
15
20
0.70
0.75
0.80
0.85
mean.p
0
4
8
12
0.50
0.55
0.60
0.65
0.70