

Abundance models with detection error

Peter Solymos and Subhash Lele

July 16, 2016 — Madison, WI — NACCB Congress

We can easily generalize this to model abundance surveys. The N-mixture model is the simplest (though unrealistic in practice).

Assumptions

- Replicate surveys,
- independence,
- closed population.

Specification of the hierarchical model

- True abundance model: $N_i \sim \text{Poisson}(\lambda)$ for locations $i = 1, 2, \dots, n$.
- Observation model: $(Y_{i,t} | N_i) \sim \text{Binomial}(N_i, p)$ for visits $t = 1, 2, \dots, T$.

```
set.seed(1234)
n <- 200
T <- 1
p <- 0.6
lambda <- 4.2
N <- rpois(n = n, lambda = lambda)
Y <- matrix(NA, n, T)
for (t in 1:T) {
  Y[,t] <- rbinom(n = n, size = N, prob = p)
}
table(N = N, Y = apply(Y, 1, max))
```

```
##      Y
## N    0  1  2  3  4  5  6  7
## 0    3  0  0  0  0  0  0  0
## 1    5  7  0  0  0  0  0  0
## 2    5 12 11  0  0  0  0  0
## 3    1 15 21  6  0  0  0  0
## 4    3  4 14 11  7  0  0  0
## 5    1  2  7 13  7  0  0  0
## 6    0  2  1  6  6  4  0  0
## 7    0  0  1  3  4  5  0  0
## 8    0  0  0  0  1  1  3  0
## 9    0  0  0  2  0  0  1  1
## 10   0  0  0  1  1  0  0  1
## 12   0  0  1  0  0  0  0  0
```

```
library(dclone)
```

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

```
## Loading required package: parallel
```

```
## Loading required package: Matrix
```

```
## dclone 2.1-1      2016-01-11
```

```
library(rjags)
```

```
## Linked to JAGS 4.0.1
```

```
## Loaded modules: basemod,bugs
```

```
model <- custommodel("model {  
  for (i in 1:n) {  
    N[i] ~ dpois(lambda)  
    for (t in 1:T) {  
      Y[i,t] ~ dbin(p, N[i])  
    }  
  }  
  p ~ dunif(0.001, 0.999)  
  lambda ~ dlnorm(0, 0.001)  
}")  
dat <- list(Y = Y, n = n, T = T)  
ini <- list(N = apply(Y, 1, max) + 1)  
fit <- jags.fit(data = dat, params = c("p", "lambda"),  
  n.update = 10000,  
  model = model, inits = ini)
```

```
## Compiling model graph  
##   Resolving undeclared variables  
##   Allocating nodes  
## Graph information:  
##   Observed stochastic nodes: 200  
##   Unobserved stochastic nodes: 202  
##   Total graph size: 408  
##  
## Initializing model
```

```
summary(fit)
```

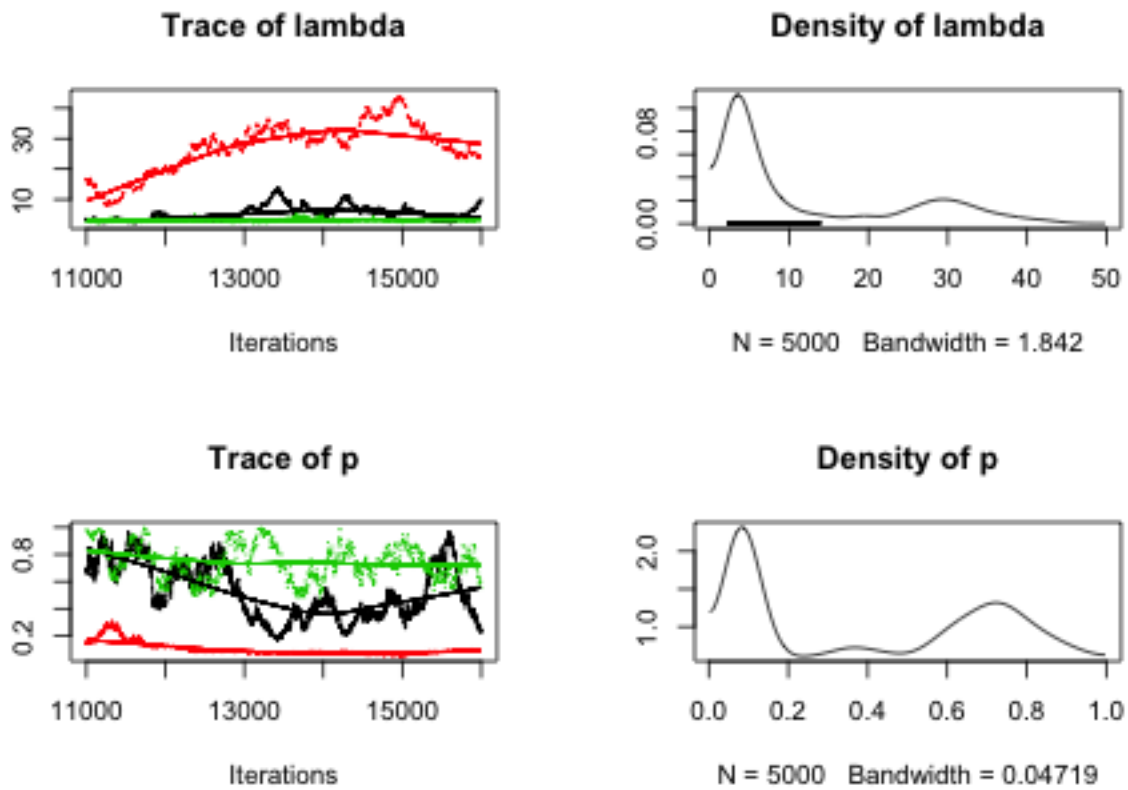
```
##  
## Iterations = 11001:16000  
## Thinning interval = 1  
## Number of chains = 3  
## Sample size per chain = 5000  
##  
## 1. Empirical mean and standard deviation for each variable,  
##   plus standard error of the mean:  
##  
##           Mean      SD Naive SE Time-series SE  
## lambda 11.9345 11.8880 0.097065      1.93754
```

```
## p      0.4568  0.3046  0.002487      0.03231
##
## 2. Quantiles for each variable:
##
##      2.5%   25%   50%   75%   97.5%
## lambda 2.44006 3.2421 5.0195 23.1315 37.9013
## p      0.06176 0.1047 0.4686  0.7284  0.9567
```

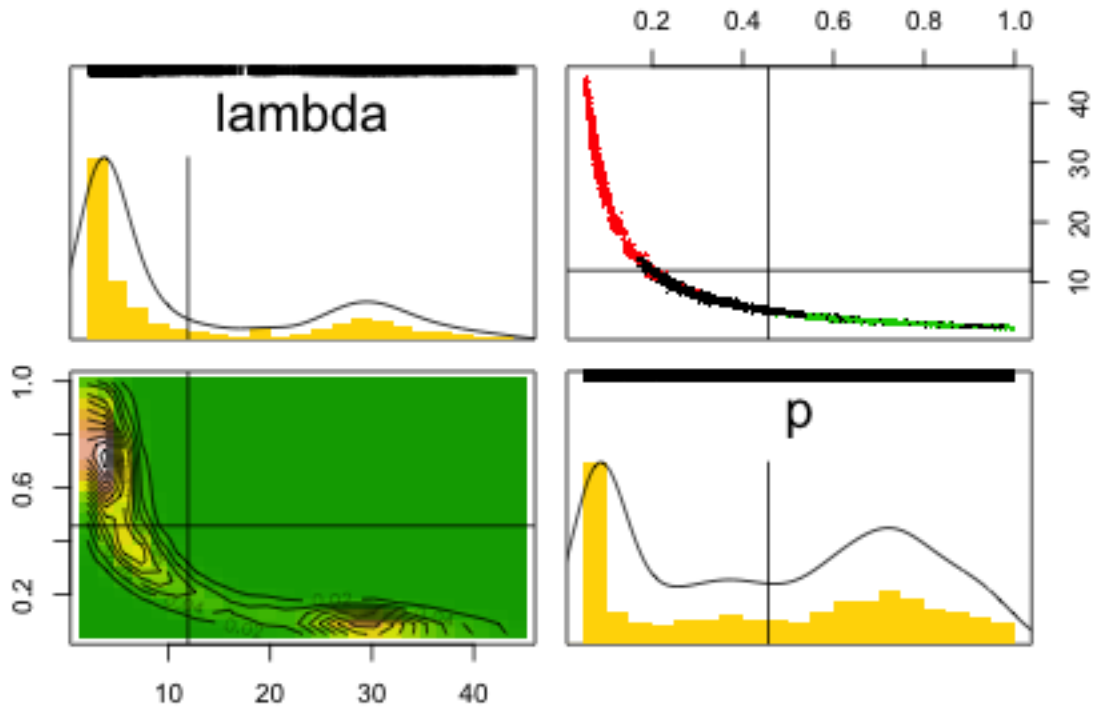
```
gelman.diag(fit)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## lambda      4.13      18.12
## p           3.46       8.66
##
## Multivariate psrf
##
## 3.74
```

```
plot(fit)
```



```
pairs(fit)
```



```
ifun <- function(model, n.clones) {  
  dclone(list(N = apply(Y, 1, max) + 1), n.clones)  
}  
dcfit <- dc.fit(data = dat,  
  params = c("p", "lambda"), model = model,  
  inits = ini, initsfun = ifun,  
  n.clones = c(1, 2, 4, 8),  
  unchanged = "T", multiply = "n")
```

```
##  
## Fitting model with 1 clone  
##  
## Compiling model graph  
##   Resolving undeclared variables  
##   Allocating nodes  
## Graph information:  
##   Observed stochastic nodes: 200  
##   Unobserved stochastic nodes: 202  
##   Total graph size: 408  
##  
## Initializing model  
##
```

```

##
## Fitting model with 2 clones
##
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 400
##   Unobserved stochastic nodes: 402
##   Total graph size: 808
##
## Initializing model
##
##
## Fitting model with 4 clones
##
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 800
##   Unobserved stochastic nodes: 802
##   Total graph size: 1608
##
## Initializing model
##
##
## Fitting model with 8 clones
##
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 1600
##   Unobserved stochastic nodes: 1602
##   Total graph size: 3208
##
## Initializing model

## Warning in dclone::.dcFit(data, params, model, inits, n.clones, multiply =
## multiply, : chains convergence problem, see R.hat values

```

```
summary(dcfit)
```

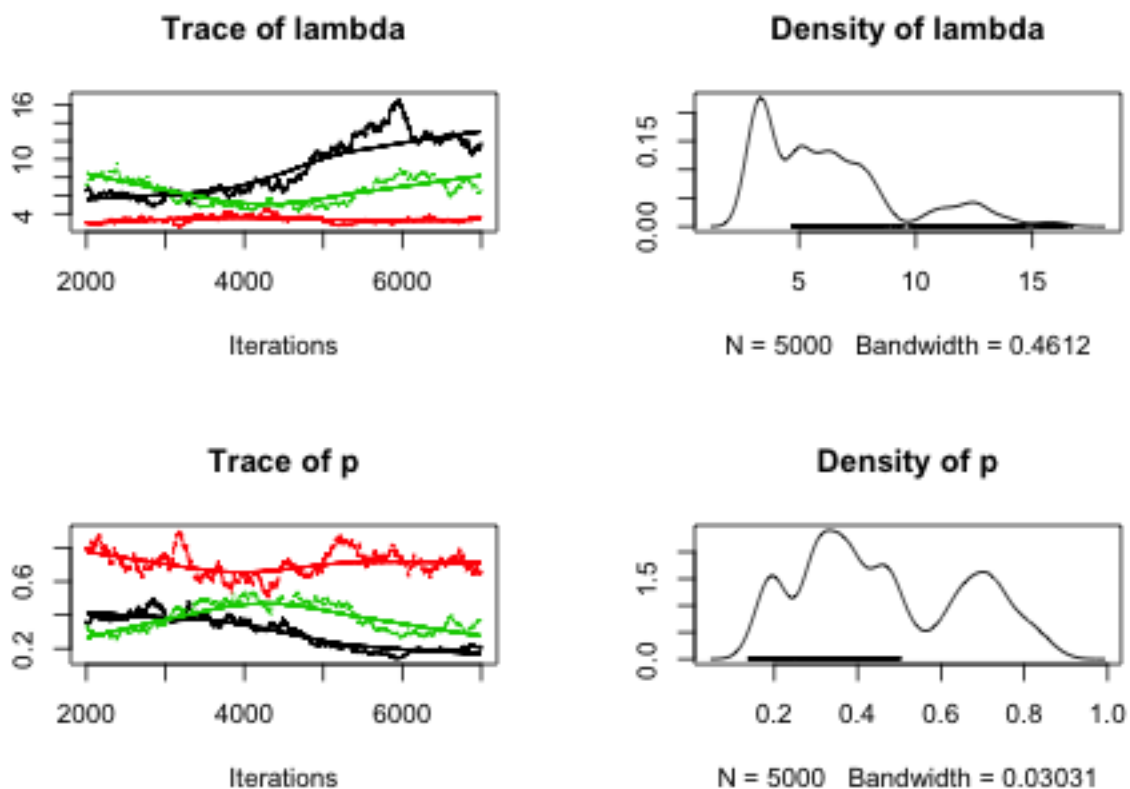
```

##
## Iterations = 2001:7000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
## Number of clones = 8
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:

```

```
##
##           Mean      SD  DC SD Naive SE Time-series SE R hat
## lambda 6.2655 3.0678 8.6771 0.025049      1.10484 2.974
## p      0.4626 0.1957 0.5534 0.001597      0.03369 4.387
##
## 2. Quantiles for each variable:
##
##           2.5%    25%    50%    75%    97.5%
## lambda 2.8570 3.6273 5.575 7.6167 13.6665
## p      0.1718 0.3097 0.423 0.6496 0.8245
```

```
plot(dcfit)
```

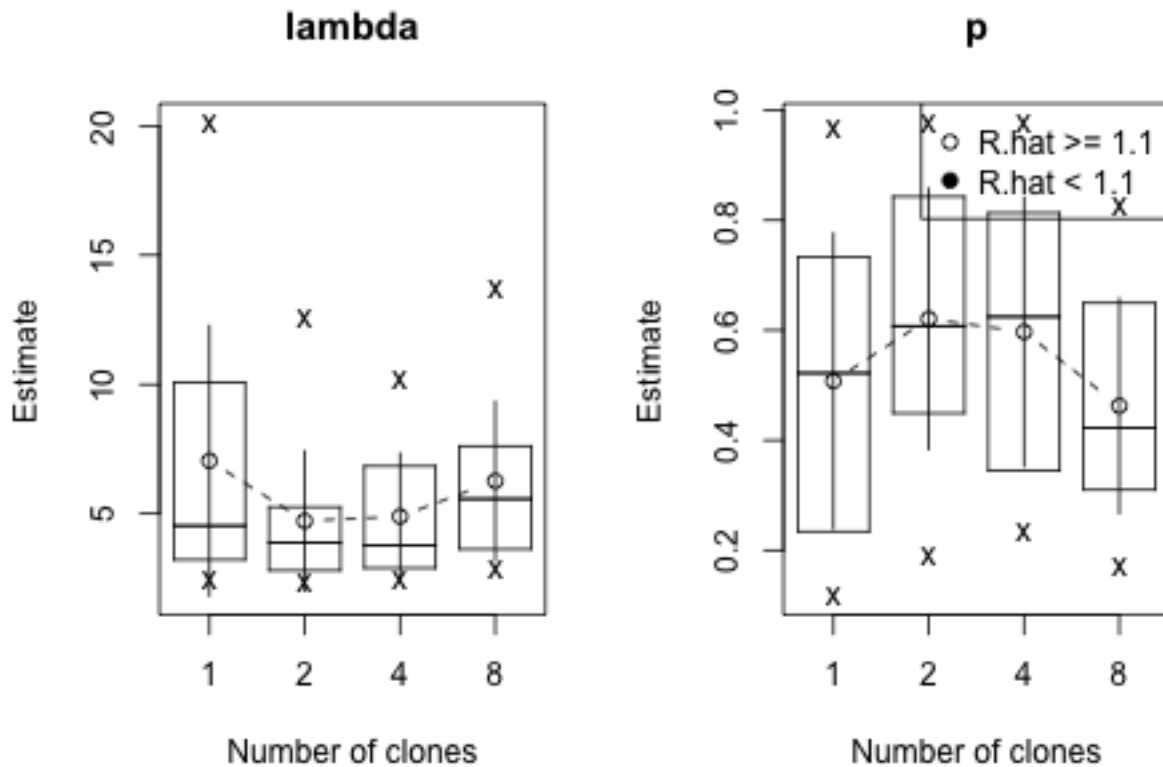


```
dctable(dcfit)
```

```
## $lambda
## n.clones mean      sd      2.5%    25%    50%    75%    97.5%
## 1         1 7.045741 5.227405 2.418524 3.218200 4.536519 10.067738 20.12716
## 2         2 4.718542 2.708924 2.389454 2.805515 3.879226 5.256660 12.55633
## 3         4 4.892891 2.445407 2.411172 2.898623 3.784789 6.840735 10.17472
## 4         8 6.265479 3.067828 2.857031 3.627332 5.574596 7.616673 13.66655
##      r.hat
## 1 1.805430
## 2 1.946222
```

```
## 3 2.337662
## 4 2.973952
##
## $p
##   n.clones      mean      sd      2.5%      25%      50%      75%
## 1         1 0.5076574 0.2686061 0.1168184 0.2341175 0.5220970 0.7335185
## 2         2 0.6208203 0.2372355 0.1890355 0.4491435 0.6078694 0.8429234
## 3         4 0.5969854 0.2446757 0.2317374 0.3445532 0.6244197 0.8134395
## 4         8 0.4625521 0.1956510 0.1717936 0.3097417 0.4229773 0.6495672
##      97.5%   r.hat
## 1 0.9657970 2.016793
## 2 0.9777050 1.924369
## 3 0.9756111 3.133583
## 4 0.8245064 4.387481
##
## attr("class")
## [1] "dctable"
```

```
plot(dctable(dcfit))
```

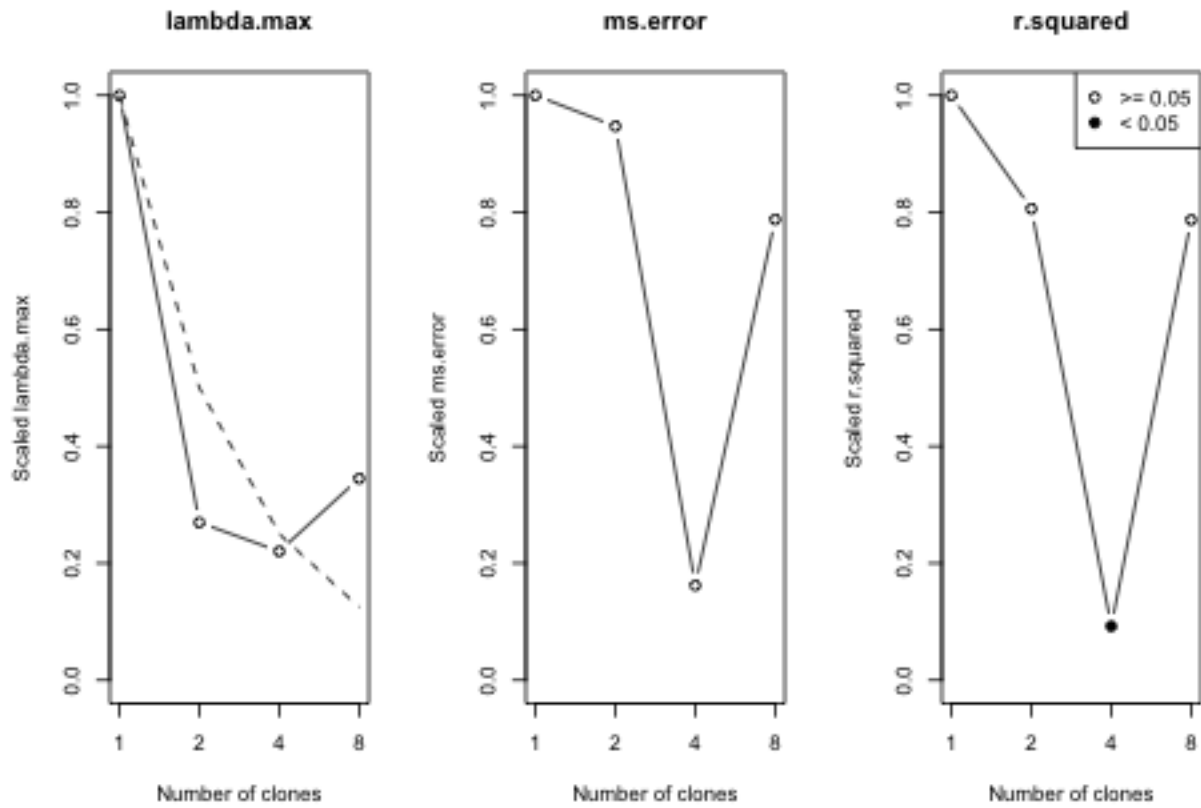


```
dcdiag(dcfit)
```

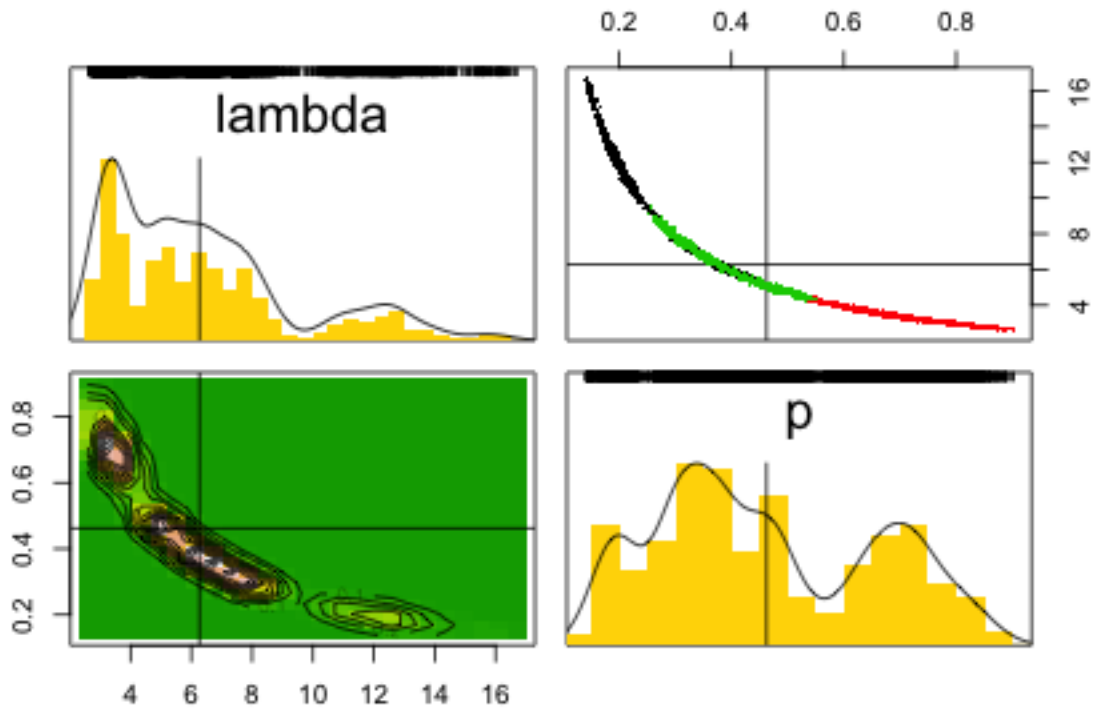
```
##   n.clones lambda.max ms.error r.squared   r.hat
## 1         1 27.379838 0.9169614 0.13153197 1.806031
```

```
## 2      2  7.382432 0.8686344 0.10603883 1.718781
## 3      4  6.032912 0.1485242 0.01213715 2.847335
## 4      8  9.442417 0.7225159 0.10352928 3.860971
```

```
plot(dcdiag(dcfit))
```



```
pairs(dcfit)
```

As before it is easy to include covariates in the models. There are various extensions and modifications proposed to this basic model. See Lele et al., Solymos et al, Solymos and Lele, Dail and Madsen. (Here we can advertise our work on single survey method and the poster.)

Single visit abundance model with covariates:

```
set.seed(1234)
n <- 200
x <- rnorm(n)
z <- rnorm(n)
beta <- c(0.9, 0.5)
theta <- c(0.8, -0.5)
Z <- model.matrix(~z)
X <- model.matrix(~x)
p <- plogis(Z %*% theta)
lambda <- exp(X %*% beta)
N <- rpois(n = n, lambda = lambda)
Y <- rbinom(n = n, size = N, prob = p)
table(N = N, Y = Y)
```

```
##      Y
## N    0  1  2  3  4  5  6  7  8 11 13
## 0   27  0  0  0  0  0  0  0  0  0  0
## 1   19 25  0  0  0  0  0  0  0  0  0
## 2    4 17 21  0  0  0  0  0  0  0  0
## 3    0  3 13 15  0  0  0  0  0  0  0
## 4    0  3  7  7  6  0  0  0  0  0  0
```

```
## 5 0 0 2 5 3 2 0 0 0 0 0
## 6 0 0 0 0 0 2 0 0 0 0 0
## 7 0 0 1 0 0 3 1 2 0 0 0
## 8 0 0 0 0 0 1 0 2 0 0 0
## 9 0 0 0 0 0 0 1 2 1 0 0
## 10 0 0 0 0 0 0 0 3 0 0 0
## 13 0 0 0 0 0 0 0 0 0 1 0
## 15 0 0 0 0 0 0 0 0 0 0 1
```

```
## naive abundance parameter estimates
m <- glm(Y ~ x, family = poisson("log"))
coef(m)
```

```
## (Intercept)          x
## 0.5473770 0.5715644
```

```
library(detect)
```

```
## Warning: package 'detect' was built under R version 3.2.4
```

```
## Loading required package: Formula
```

```
## Loading required package: stats4
```

```
## Loading required package: pbapply
```

```
## Warning: package 'pbapply' was built under R version 3.2.5
```

```
## detect 0.4-0      2016-03-02
```

```
md <- svabu(Y ~ x | z, zeroinfl = FALSE)
coef(md)
```

```
## sta_(Intercept)      sta_x det_(Intercept)      det_z
## 0.6364075 0.5712775 3.4375548 -1.6490555
```

```
model <- custommodel("model {
  for (i in 1:n) {
    N[i] ~ dpois(lambda[i])
    Y[i] ~ dbin(p[i], N[i])
    log(lambda[i]) <- inprod(X[i,], beta)
    logit(p[i]) <- inprod(Z[i,], theta)
  }
  for (j in 1:px) {
    beta[j] ~ dnorm(naive[j], 0.1)
  }
  for (j in 1:pz) {
    theta[j] ~ dnorm(0, 0.01)
  }
}")
```

```

dat <- list(Y = Y, n = n, X = X, Z = Z,
  px = ncol(X), pz = ncol(Z), naive = coef(m))
ini <- list(N = Y + 1)
fit <- jags.fit(data = dat, params = c("beta", "theta"),
  n.update = 5000,
  model = model, inits = ini)

```

```

## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 200
##   Unobserved stochastic nodes: 204
##   Total graph size: 2417
##
## Initializing model

```

```

## DC
ifun <- function(model, n.clones) {
  dclone(list(N = Y + 1), n.clones)
}
dcfit <- dc.fit(data = dat,
  params = c("beta", "theta"), model = model,
  inits = ini, initsfun = ifun,
  n.clones = c(1, 2, 4, 8),
  #   n.update = 5000,
  unchanged = c("px", "pz", "naive"), multiply = "n")

```

```

##
## Fitting model with 1 clone
##
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 200
##   Unobserved stochastic nodes: 204
##   Total graph size: 2417
##
## Initializing model
##
## Fitting model with 2 clones
##
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 400
##   Unobserved stochastic nodes: 404
##   Total graph size: 4017
##
## Initializing model

```

```

##
##
## Fitting model with 4 clones
##
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 800
##   Unobserved stochastic nodes: 804
##   Total graph size: 7217
##
## Initializing model
##
##
## Fitting model with 8 clones
##
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 1600
##   Unobserved stochastic nodes: 1604
##   Total graph size: 13617
##
## Initializing model

```

```
summary(dcfits)
```

```

##
## Iterations = 2001:7000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
## Number of clones = 8
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean      SD DC SD Naive SE Time-series SE R hat
## beta[1]  0.6399 0.03590 0.1015 0.0002931      0.0034841 1.015
## beta[2]  0.5716 0.01616 0.0457 0.0001319      0.0002244 1.001
## theta[1] 3.5078 0.77234 2.1845 0.0063061      0.1176666 1.019
## theta[2] -1.6721 0.36895 1.0435 0.0030124      0.0543331 1.016
##
## 2. Quantiles for each variable:
##
##           2.5%    25%    50%    75%    97.5%
## beta[1]  0.5779 0.6145 0.6367 0.6628 0.7189
## beta[2]  0.5398 0.5607 0.5716 0.5824 0.6033
## theta[1] 2.2261 2.8970 3.4633 4.0228 5.0853
## theta[2] -2.4232 -1.9244 -1.6525 -1.3907 -1.0355

```

```
dcdiag(dcfit)
```

```
##   n.clones lambda.max   ms.error  r.squared   r.hat
## 1         1 44.7174344 16.3539951 0.31932661 1.170655
## 2         2 21.5394647 13.3391346 0.27904243 1.200791
## 3         4  1.6466965  1.7234011 0.08052065 1.024228
## 4         8  0.7267786  0.3135922 0.02513831 1.014386
```

Learning with DC

```
model <- custommodel("model {
  for (i in 1:n) {
    N[i] ~ dpois(lambda[i])
    Y[i] ~ dbin(p[i], N[i])
    log(lambda[i]) <- inprod(X[i,], beta)
    logit(p[i]) <- inprod(Z[i,], theta)
  }

  cf[1:(px + pz)] ~ dmnorm(pr[,1], pr[,2:(px + pz + 1)])
  beta <- cf[1:px]
  theta <- cf[(px + 1):(px + pz)]
}")
dat <- list(Y = Y, n = n, X = X, Z = Z,
  px = ncol(X), pz = ncol(Z),
  pr = unname(cbind(c(coef(m), rep(0, ncol(Z))),
    diag(0.01, ncol(X) + ncol(Z))))))
ini <- list(N = Y + 1)
ifun <- function(model, n.clones) {
  dclone(list(N = Y + 1), n.clones)
}
## function to update prior
## defined as Multivariate Normal distribution
ufun <- function(model, n.clones) {
  cbind(coef(model), solve(vcov(model)))
}
dcfit <- dc.fit(data = dat,
  params = c("beta", "theta"), model = model,
  inits = ini, initsfun = ifun,
  update = "pr", updatefun = ufun,
  n.clones = c(1, 2, 4, 8),
  n.update = 5000,
  unchanged = c("px", "pz", "pr"), multiply = "n")
```

```
##
## Fitting model with 1 clone
##
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 200
```

```

## Unobserved stochastic nodes: 201
## Total graph size: 2428
##
## Initializing model
##
##
## Fitting model with 2 clones
##
## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
## Observed stochastic nodes: 400
## Unobserved stochastic nodes: 401
## Total graph size: 4028
##
## Initializing model
##
##
## Fitting model with 4 clones
##
## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
## Observed stochastic nodes: 800
## Unobserved stochastic nodes: 801
## Total graph size: 7228
##
## Initializing model
##
##
## Fitting model with 8 clones
##
## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
## Observed stochastic nodes: 1600
## Unobserved stochastic nodes: 1601
## Total graph size: 13628
##
## Initializing model

## Warning in dclone::.dcFit(data, params, model, inits, n.clones, multiply =
## multiply, : chains convergence problem, see R.hat values

```

```
summary(dcfrit)
```

```

##
## Iterations = 6001:11000
## Thinning interval = 1
## Number of chains = 3

```

```

## Sample size per chain = 5000
## Number of clones = 8
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean      SD   DC SD   Naive SE Time-series SE R hat
## beta[1]  0.7159 0.05312 0.15025 0.0004337      0.0065051 1.824
## beta[2]  0.5681 0.01621 0.04586 0.0001324      0.0004449 1.003
## theta[1] 2.1994 0.45697 1.29250 0.0037311      0.1069388 2.086
## theta[2] -1.0494 0.21811 0.61692 0.0017809      0.0336612 1.866
##
## 2. Quantiles for each variable:
##
##           2.5%    25%    50%    75%    97.5%
## beta[1]  0.6371 0.6770 0.7046 0.7470 0.8368
## beta[2]  0.5361 0.5572 0.5685 0.5791 0.5988
## theta[1] 1.3391 1.8951 2.2535 2.5801 2.9284
## theta[2] -1.4289 -1.2190 -1.0572 -0.9010 -0.6427

```

```
dcdiag(dcfits)
```

```

##   n.clones lambda.max  ms.error  r.squared  r.hat
## 1         1  5.5157831 0.57938641 0.029820352 3.584440
## 2         2  2.4999785 0.48352025 0.035814958 1.973305
## 3         4  0.4348400 0.02239688 0.002508366 1.184487
## 4         8  0.2533176 0.02424999 0.002914900 1.732422

```

Zero-inflated Poisson latent process

This really becomes an issue when $T = 1$. With $T > 1$ it is much easier to distinguish non occupied ($O_i = 0$ or $N_i = 0|O_i = 1$) locations when all the detection history is 0, and non-detections when some of the detection history is >0 if p is not too small.

```

set.seed(1234)
n <- 100
T <- 2
p <- 0.6
lambda <- 3.5
q <- 0.25
O <- rbinom(n, size = 1, prob = q)
N <- O * rpois(n = n, lambda = lambda)
Y <- matrix(NA, n, T)
for (t in 1:T) {
  Y[,t] <- rbinom(n = n, size = N, prob = p)
}
table(N = N, Y = apply(Y, 1, max))

```

```

##      Y
## N    0 1 2 3 4 5 6 7
## 0  82 0 0 0 0 0 0 0
## 1   0 2 0 0 0 0 0 0

```

```
## 2 0 0 1 0 0 0 0 0
## 3 0 1 2 3 0 0 0 0
## 4 0 0 1 3 0 0 0 0
## 5 0 0 0 1 0 0 0 0
## 6 0 0 0 0 1 1 0 0
## 9 0 0 0 0 0 0 0 1
## 10 0 0 0 0 0 0 1 0
```

```
library(dclone)
model <- custommodel("model {
  for (i in 1:n) {
    O[i] ~ dbern(q)
    N[i] ~ dpois(lambda * O[i])
    for (t in 1:T) {
      Y[i,t] ~ dbin(p, N[i])
    }
  }
  p ~ dunif(0.001, 0.999)
  lambda ~ dlnorm(0, 0.001)
  q ~ dunif(0.001, 0.999)
}")
dat <- list(Y = Y, n = n, T = T)
## initial values are trickier
ini <- list(N = ifelse(rowSums(Y) > 0, 1, 0) * (apply(Y, 1, max) + 1),
  O = ifelse(rowSums(Y) > 0, 1, 0))
fit <- jags.fit(data = dat, params = c("p", "lambda", "q"),
  n.update = 10000,
  model = model, inits = ini)
```

```
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 200
##   Unobserved stochastic nodes: 203
##   Total graph size: 511
##
## Initializing model
```

```
summary(fit)
```

```
##
## Iterations = 11001:16000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##   plus standard error of the mean:
##
##           Mean      SD Naive SE Time-series SE
## lambda 3.9870 1.11639 0.0091153 0.0576736
## p       0.6303 0.11541 0.0009423 0.0056808
```

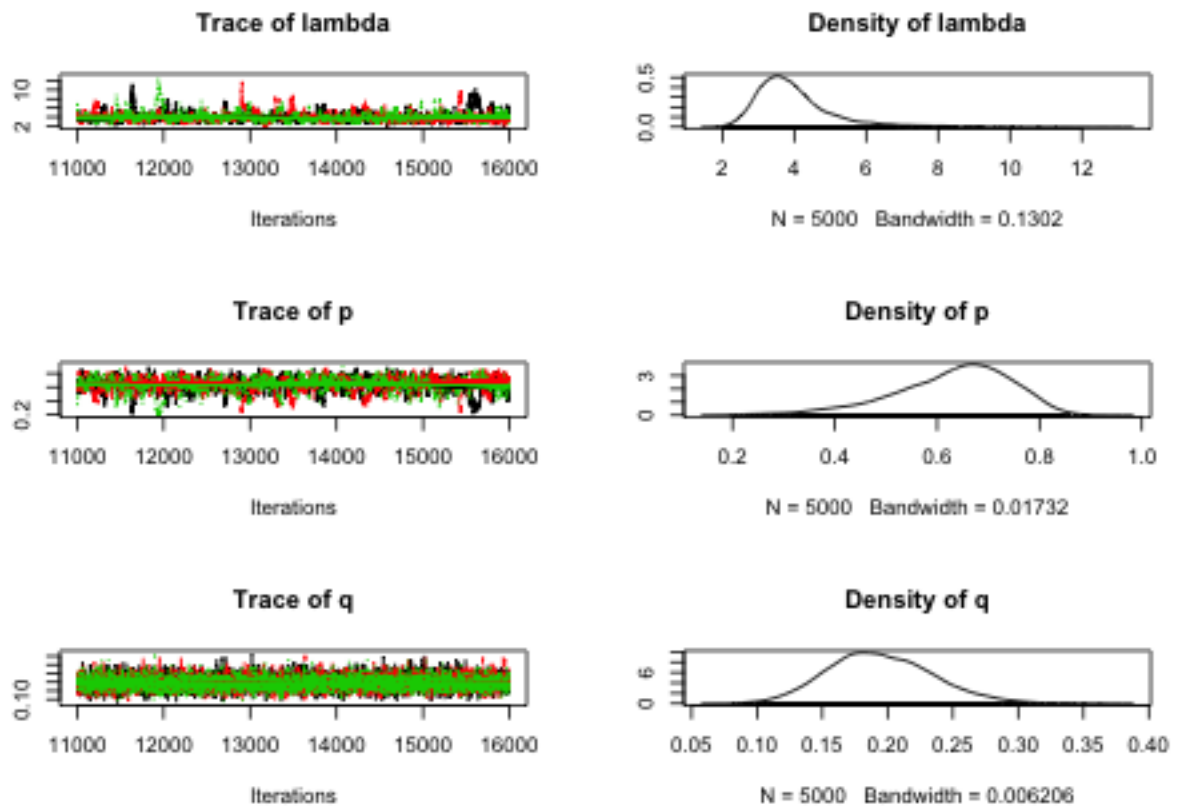


```
## q      0.1948 0.04006 0.0003271      0.0004602
##
## 2. Quantiles for each variable:
##
##      2.5%   25%   50%   75%  97.5%
## lambda 2.5479 3.2685 3.7626 4.3952 6.9094
## p      0.3614 0.5625 0.6472 0.7123 0.8111
## q      0.1227 0.1669 0.1925 0.2209 0.2787
```

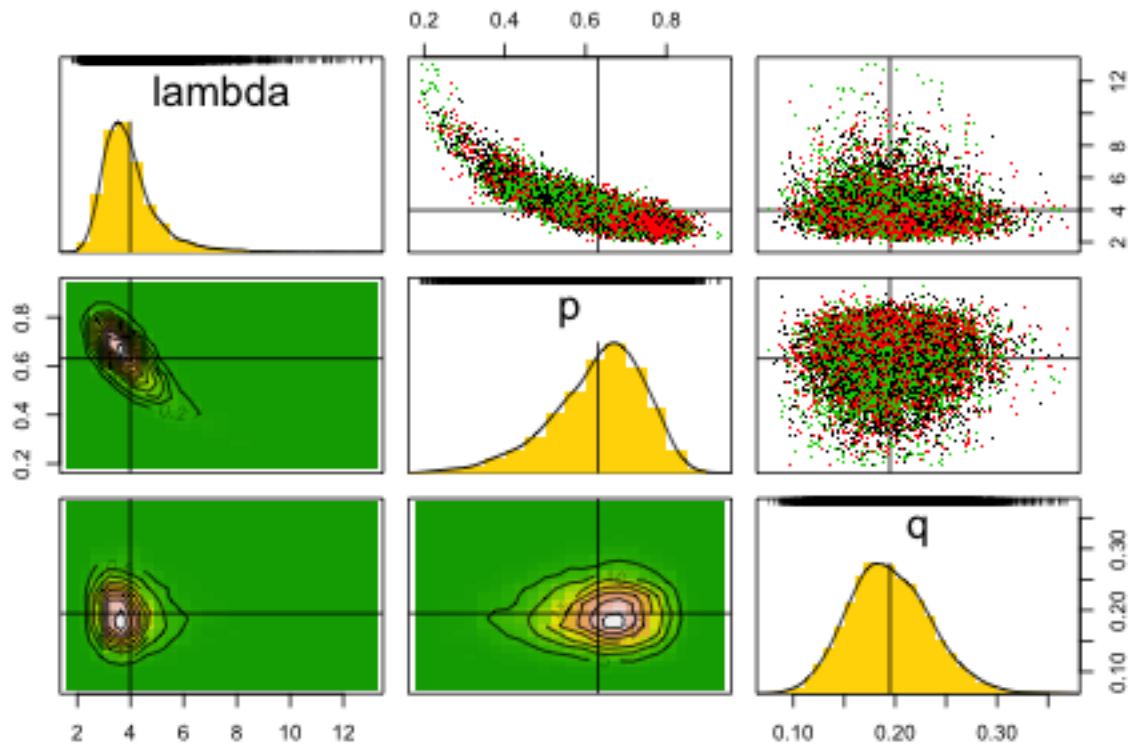
```
gelman.diag(fit)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## lambda      1.01      1.01
## p           1.00      1.01
## q           1.00      1.00
##
## Multivariate psrf
##
## 1
```

```
plot(fit)
```



```
pairs(fit)
```



Data cloning for zero inflated data: issues might arise with parent values, that is why we do conditional likelihood estimation. It is also possible to use data cloning and JAGS for conditional likelihood estimation as explained in Solymos et al. 2012 ([PDF](#)).

Poisson-Poisson mixture

This modification can be suitable in cases when there is e.g. double counting of individuals, or false positives.

```
set.seed(1234)
n <- 200
x <- rnorm(n)
z <- rnorm(n)
beta <- c(0.9, 0.5)
theta <- c(0.8, -0.5)
Z <- model.matrix(~z)
X <- model.matrix(~x)
p <- plogis(Z %*% theta)
lambda <- exp(X %*% beta)
N <- rpois(n = n, lambda = lambda)
Y <- rpois(n = n, lambda = p * N)
table(N = N, Y = Y)
```

```

##      Y
## N    0  1  2  3  4  5  6  7  8 10
## 0  27  0  0  0  0  0  0  0  0
## 1  26 10  4  4  0  0  0  0  0
## 2  15 11 11  4  1  0  0  0  0
## 3   4 12  6  6  2  1  0  0  0
## 4   2  6  3  4  5  2  1  0  0
## 5   1  2  5  0  2  2  0  0  0
## 6   0  0  1  0  1  0  0  0  0
## 7   0  2  0  0  3  0  0  2  0
## 8   0  0  0  1  1  1  0  0  0
## 9   0  1  0  0  0  1  0  1  1
## 10  0  0  0  0  0  0  1  0  1
## 13  0  0  0  0  0  0  0  0  1
## 15  0  0  0  0  0  0  0  1  0

```

```

m <- glm(Y ~ x, family = poisson("log"))

## N is of interest for e.g. prediction
modell1 <- custommodel("model {
  for (i in 1:n) {
    N[i] ~ dpois(lambda[i])
    #Y[i] ~ dbin(p[i], N[i])
    Y[i] ~ dpois(p[i] * N[i]) # this is the change
    log(lambda[i]) <- inprod(X[i,], beta)
    logit(p[i]) <- inprod(Z[i,], theta)
  }
  for (j in 1:px) {
    beta[j] ~ dnorm(naive[j], 0.1)
  }
  for (j in 1:pz) {
    theta[j] ~ dnorm(0, 0.01)
  }
}")

## more efficient when N is not of interest
modell2 <- custommodel("model {
  for (i in 1:n) {
    Y[i] ~ dpois(p[i] * lambda[i])
    log(lambda[i]) <- inprod(X[i,], beta)
    logit(p[i]) <- inprod(Z[i,], theta)
  }
  for (j in 1:px) {
    beta[j] ~ dnorm(naive[j], 0.1)
  }
  for (j in 1:pz) {
    theta[j] ~ dnorm(0, 0.01)
  }
}")

dat <- list(Y = Y, n = n, X = X, Z = Z,
  px = ncol(X), pz = ncol(Z), naive = coef(m))
ini <- list(N = Y + 1)
fit1 <- jags.fit(data = dat, params = c("beta", "theta"),
  n.update = 5000, model = modell1, inits = ini)

```

```
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 200
##   Unobserved stochastic nodes: 204
##   Total graph size: 2617
##
## Initializing model
```

```
fit2 <- jags.fit(data = dat, params = c("beta", "theta"),
  n.update = 5000, model = model2)
```

```
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 200
##   Unobserved stochastic nodes: 4
##   Total graph size: 2417
##
## Initializing model
```

```
coef(fit1)
```

```
##   beta[1]   beta[2]   theta[1]   theta[2]
## 0.5090397 0.5679768 7.2596160 -1.7092087
```

```
coef(fit2)
```

```
##   beta[1]   beta[2]   theta[1]   theta[2]
## 0.5895744 0.5717896 6.3215477 -1.9587146
```

```
## DC
ifun <- function(model, n.clones) {
  dclone(list(N = Y + 1), n.clones)
}
dcfit1 <- dc.fit(data = dat,
  params = c("beta", "theta"), model = model1,
  inits = ini, initsfun = ifun,
  n.clones = c(1, 2, 4, 8),
  #   n.update = 5000,
  unchanged = c("px", "pz", "naive"), multiply = "n")
```

```
##
## Fitting model with 1 clone
##
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
```

```

## Observed stochastic nodes: 200
## Unobserved stochastic nodes: 204
## Total graph size: 2617
##
## Initializing model
##
##
## Fitting model with 2 clones
##
## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
## Observed stochastic nodes: 400
## Unobserved stochastic nodes: 404
## Total graph size: 4417
##
## Initializing model
##
##
## Fitting model with 4 clones
##
## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
## Observed stochastic nodes: 800
## Unobserved stochastic nodes: 804
## Total graph size: 8017
##
## Initializing model
##
##
## Fitting model with 8 clones
##
## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
## Observed stochastic nodes: 1600
## Unobserved stochastic nodes: 1604
## Total graph size: 15217
##
## Initializing model

dcfit2 <- dc.fit(data = dat,
  params = c("beta", "theta"), model = model2,
  n.clones = c(1, 2, 4, 8),
  # n.update = 5000,
  unchanged = c("px", "pz", "naive"), multiply = "n")

##
## Fitting model with 1 clone
##

```

```

## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 200
##   Unobserved stochastic nodes: 4
##   Total graph size: 2417
##
## Initializing model
##
##
## Fitting model with 2 clones
##
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 400
##   Unobserved stochastic nodes: 4
##   Total graph size: 3817
##
## Initializing model
##
##
## Fitting model with 4 clones
##
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 800
##   Unobserved stochastic nodes: 4
##   Total graph size: 6617
##
## Initializing model
##
##
## Fitting model with 8 clones
##
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 1600
##   Unobserved stochastic nodes: 4
##   Total graph size: 12217
##
## Initializing model

## Warning in dclone::.dcFit(data, params, model, inits, n.clones, multiply =
## multiply, : chains convergence problem, see R.hat values

```

```
coef(dcf1t1)
```

```
##      beta[1]      beta[2]      theta[1]      theta[2]  
## 0.7416785 0.5768555 0.8943633 -0.5343544
```

```
coef(dcf1t2)
```

```
##      beta[1]      beta[2]      theta[1]      theta[2]  
## 1.5825361 0.5706308 -0.6929124 -0.3035158
```