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

```
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
```

```r
library(rjags)
```

```
## Linked to JAGS 4.0.1

## Loaded modules: basemod,bugs
```

```r
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
```

```r
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 7.6540 5.9750 0.048785        2.50278
```
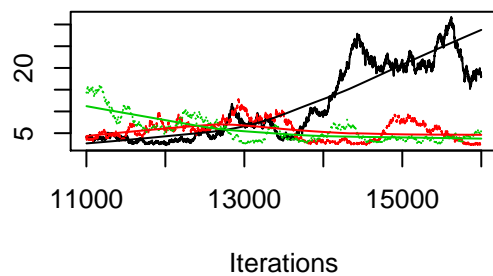
```
## p       0.4538 0.2309 0.001885        0.06876
##
## 2. Quantiles for each variable:
##
##             2.5%     25%     50%     75%   97.5%
## lambda 2.61253 3.7044 5.6807 8.4672 24.8760
## p      0.09532 0.2789 0.4155 0.6374  0.8992
```

```r
gelman.diag(fit)
```
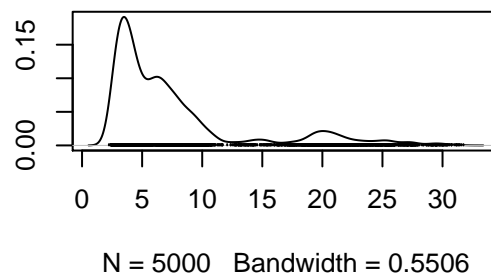
```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## lambda         1.46       3.27
## p              1.10       1.28
##
## Multivariate psrf
##
## 1.32
```
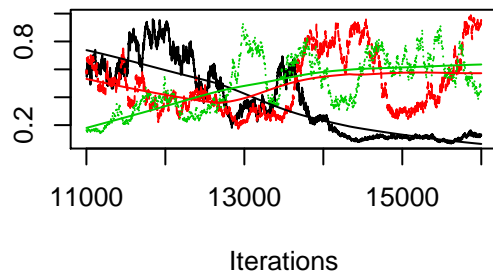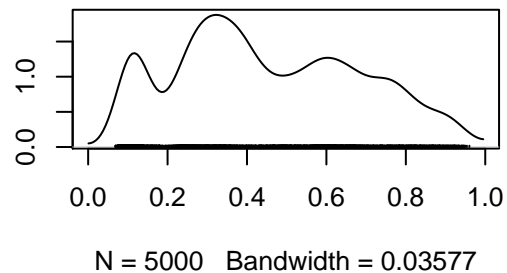
```r
plot(fit)
```
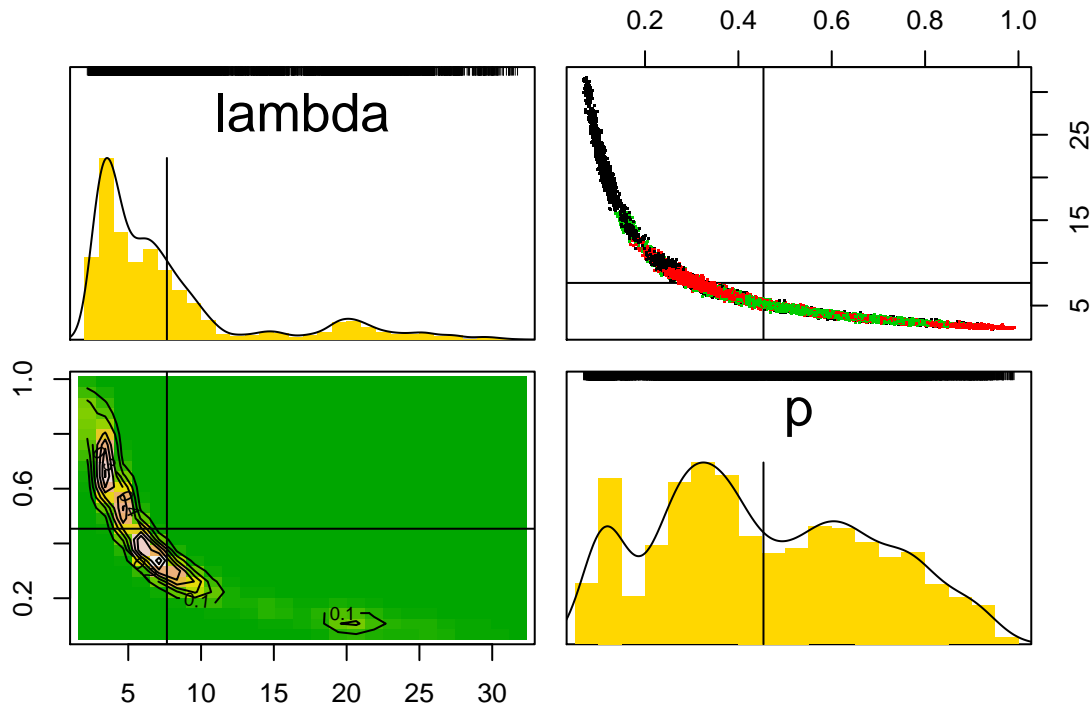

Trace of lambda

Density of lambda

Iterations

N = 5000   Bandwidth = 0.5506

Trace of p

Density of p

Iterations

N = 5000   Bandwidth = 0.03577

```r
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
```
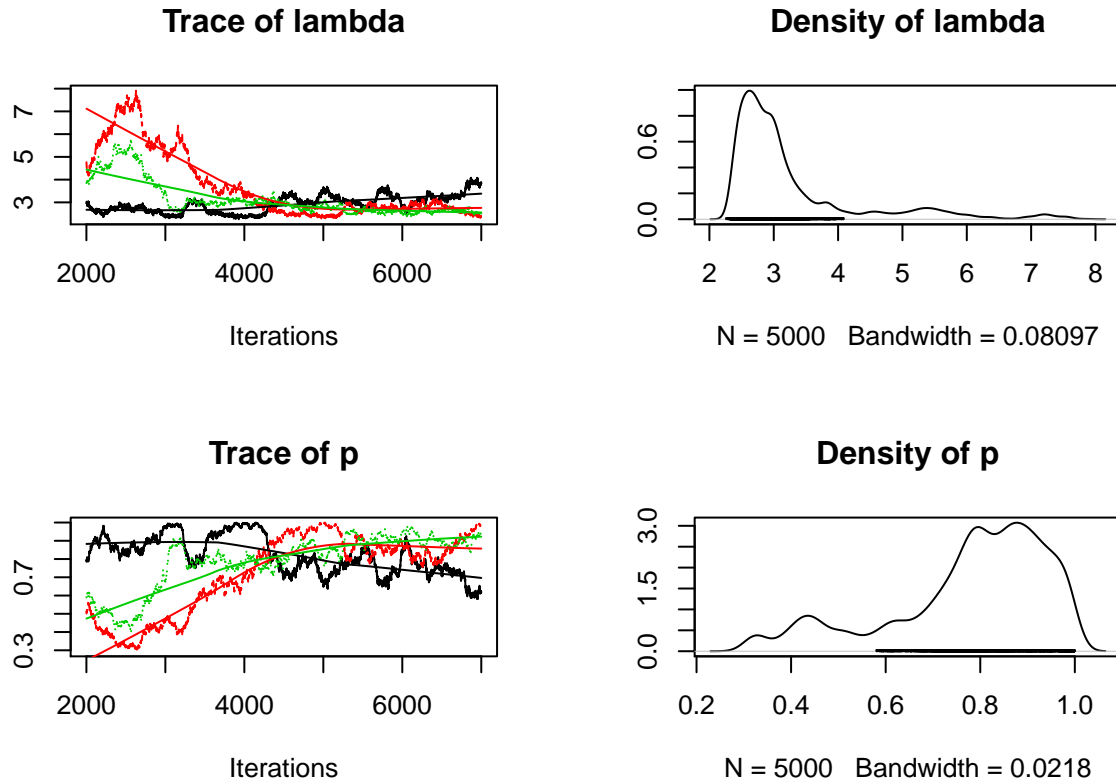
summary(dcfit)

```
##
## 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 3.2529 1.0449 2.9556 0.008532        0.51887 1.135
## p      0.7769 0.1678 0.4745 0.001370        0.07081 1.118
##
## 2. Quantiles for each variable:
##
##            2.5%    25%     50%    75%  97.5%
```

```
## lambda 2.3805 2.6220 2.8982 3.3224 6.3762
## p      0.3713 0.7116 0.8136 0.9001 0.9882
```

```
plot(dcfit)
```

Trace of lambda   **Density of lambda**

Trace of p   **Density of p**

```
dctable(dcfit)
```

```
## $lambda
##   n.clones     mean        sd      2.5%       25%       50%       75%
## 1        1 4.335856 2.3210717 2.323509 2.714756 3.574809 5.038219
## 2        2 6.269918 2.1869068 2.597268 4.909443 5.830516 7.376492
## 3        4 3.372709 0.9916788 2.346224 2.588498 3.055035 3.926600
## 4        8 3.252867 1.0449478 2.380466 2.621997 2.898224 3.322357
##       97.5%    r.hat
## 1 11.797043 1.329473
## 2 11.288124 2.320199
## 3  5.660113 1.410589
## 4  6.376250 1.134540
##
## $p
##   n.clones      mean        sd      2.5%       25%       50%       75%
## 1        1 0.6542215 0.2330751 0.2006435 0.4680436 0.6611977 0.8694708
## 2        2 0.4260455 0.1610472 0.2079487 0.3196328 0.4054266 0.4806083
## 3        4 0.7494145 0.1789487 0.4168617 0.6007671 0.7756604 0.9107091
## 4        8 0.7768742 0.1677714 0.3713303 0.7115703 0.8136430 0.9001329
##       97.5%    r.hat
## 1 0.9852292 1.335251
```

```
## 2 0.9152155 2.014152
## 3 0.9935917 1.555757
## 4 0.9882194 1.118366
##
## attr(,"class")
## [1] "dctable"
```
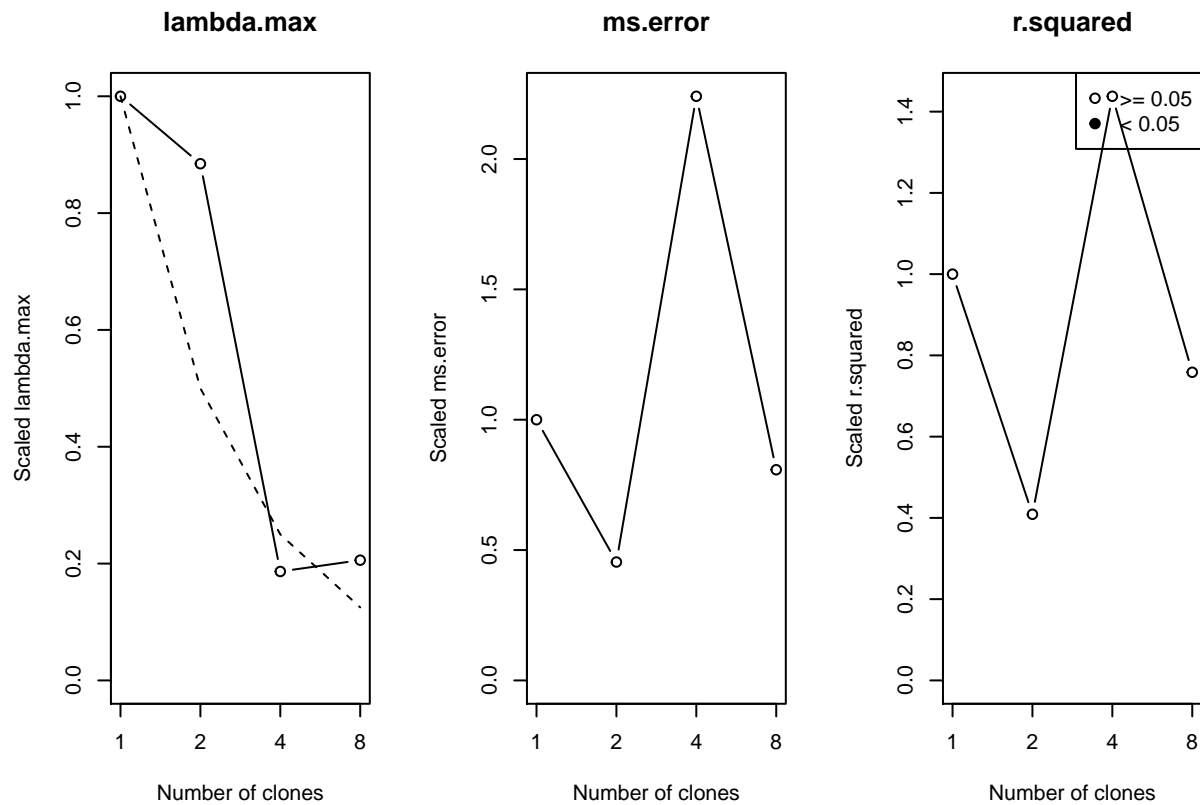
```
plot(dctable(dcfit))
```



```
dcdiag(dcfit)
```
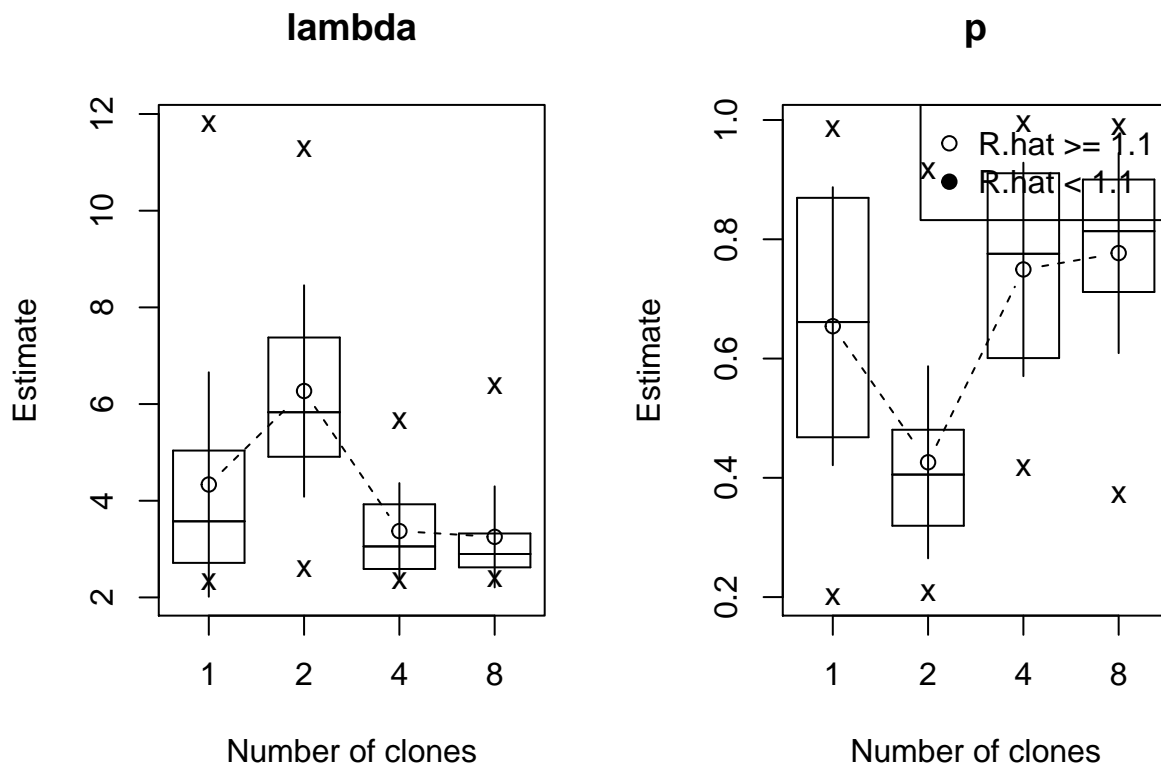
```
##   n.clones lambda.max  ms.error r.squared    r.hat
## 1        1   5.429850  7.227009 0.3552316 1.281939
## 2        2   4.802801  3.281601 0.1452522 2.019372
## 3        4   1.012300 16.193899 0.5107510 1.534705
## 4        8   1.117573  5.840285 0.2694265 1.104910
```

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

```
pairs(dcfit)
```
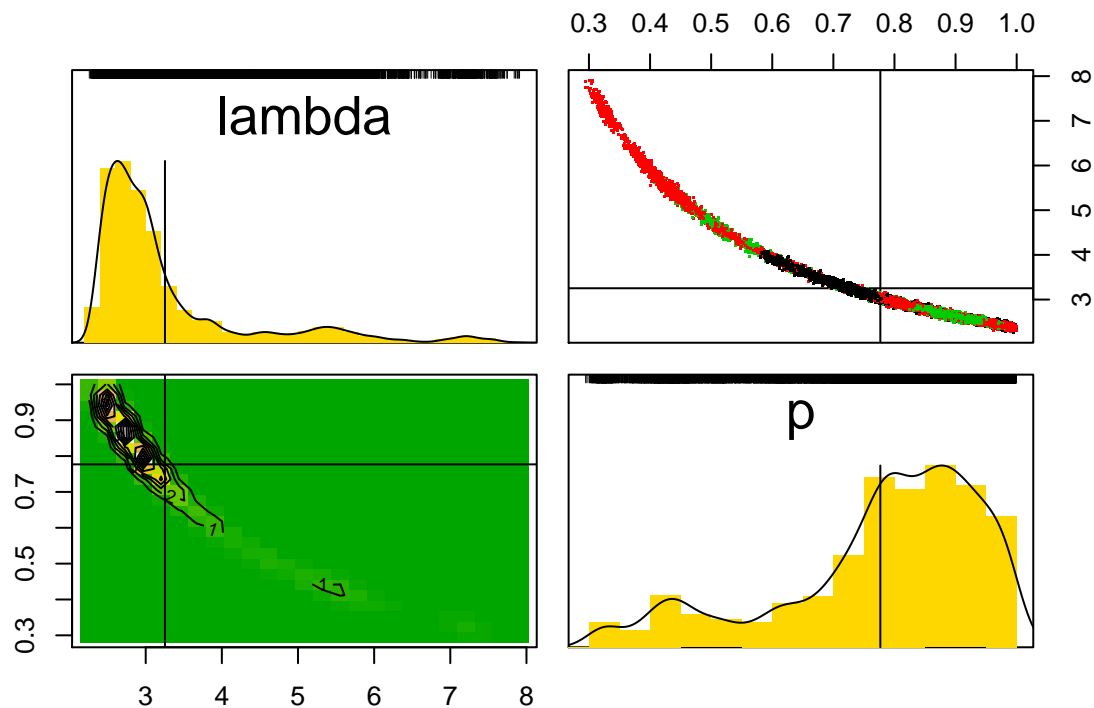

As before it is easy to include covariates in the models. There are various extensions and modifications proposed to this basic model. See Lele et al., Solymos et al, Solymos and Lele, Dail and Madsen. (Here we can advertise our work on single survey method and the poster.)

Single visit abundance model with covariates:

```r
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
```

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

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

```r
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
```

```r
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
## beta[1]    0.6337 0.03509 0.09925 0.0002865      0.0036525 1.006
## beta[2]    0.5716 0.01580 0.04470 0.0001290      0.0002106 1.000
## theta[1]   3.7623 1.02202 2.89070 0.0083447      0.1865302 1.019
## theta[2] -1.7884 0.48470 1.37093 0.0039575      0.0834242 1.021
##
## 2. Quantiles for each variable:
##
##               2.5%     25%     50%     75%    97.5%
## beta[1]    0.5721  0.6091  0.6316  0.6558  0.7102
## beta[2]    0.5404  0.5609  0.5716  0.5824  0.6025
## theta[1]   2.2758  3.0770  3.5657  4.2713  6.4444
## theta[2] -3.0210 -2.0371 -1.7061 -1.4629 -1.0657
```

```
dcdiag(dcfit)
```

```
##   n.clones lambda.max ms.error  r.squared      r.hat
## 1        1  31.310220 7.577533 0.15198717 1.031245
## 2        2  29.254473 2.237111 0.07406182 1.220126
## 3        4  37.499893 1.479262 0.10437514 2.033224
## 4        8   1.273474 1.903269 0.08268187 1.005759
```

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(dcfit)

```
##
## 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]    0.6619 0.05495 0.1554 0.0004487      0.0054983 2.372
## beta[2]    0.5706 0.01605 0.0454 0.0001311      0.0004132 1.005
## theta[1]   3.2184 1.11559 3.1554 0.0091087      0.1401412 4.673
## theta[2] -1.5318 0.52565 1.4868 0.0042919      0.0481178 4.275
##
## 2. Quantiles for each variable:
##
##              2.5%     25%     50%     75%   97.5%
## beta[1]    0.5747  0.618  0.6590  0.6979  0.7914
## beta[2]    0.5391  0.560  0.5702  0.5817  0.6012
## theta[1]   1.5484  2.399  2.8939  4.3561  5.2145
## theta[2] -2.4810 -2.086 -1.3617 -1.1503 -0.7892
```

dcdiag(dcfit)

```
##   n.clones lambda.max    ms.error   r.squared     r.hat
## 1        1 12.1181282 5.17053973 0.095522879 1.321617
## 2        2  1.3147131 0.35358204 0.013453009 1.235215
## 3        4  0.3616931 0.04321356 0.001146301 1.311300
## 4        8  1.5164742 0.06424935 0.007924711 3.780860
```

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.

```r
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
```

```r
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
```
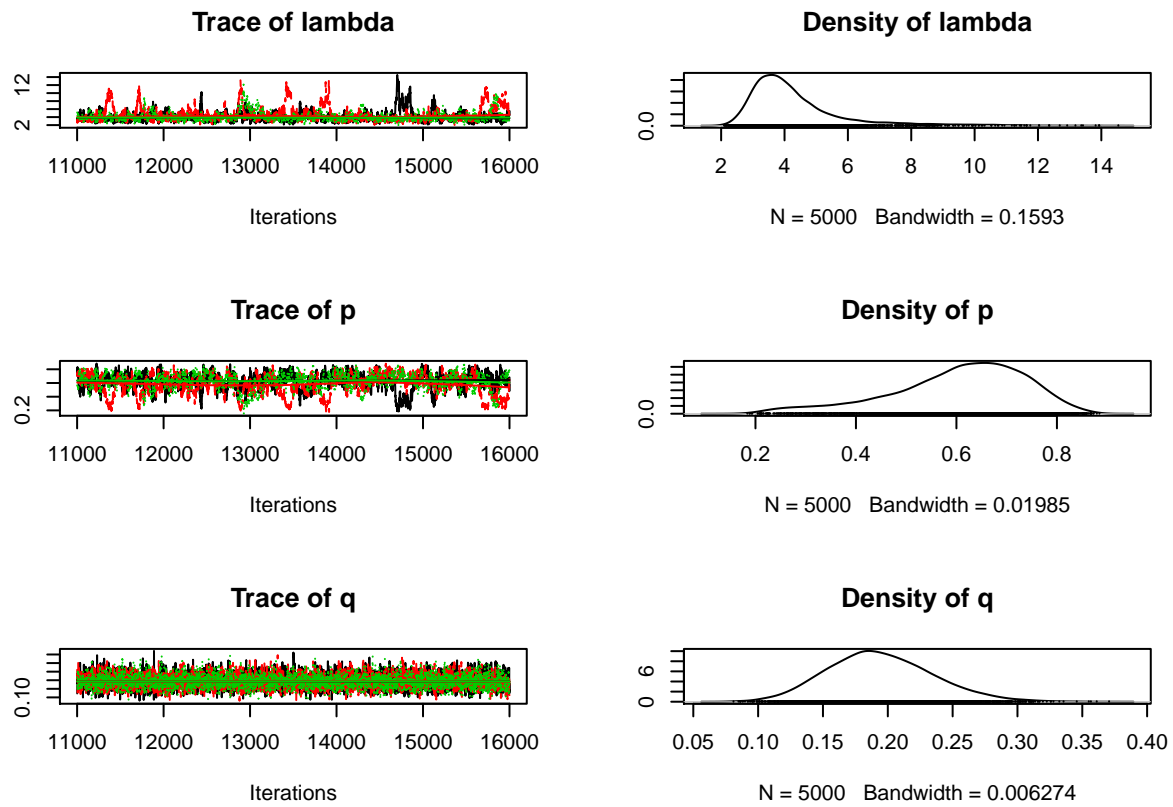
```r
summary(fit)
```

```
##
## 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 4.3021 1.5211 0.0124195      0.1114799
## p      0.6038 0.1340 0.0010945      0.0082015
## q      0.1943 0.0405 0.0003307      0.0004742
##
## 2. Quantiles for each variable:
##
##           2.5%    25%    50%    75% 97.5%
## lambda 2.5928 3.3473 3.9135 4.7251 8.878
## p      0.2781 0.5298 0.6238 0.7015 0.811
## q      0.1213 0.1661 0.1918 0.2205 0.279
```

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

```
## Potential scale reduction factors:
##
##        Point est. Upper C.I.
## lambda       1.07       1.18
## p            1.04       1.10
## q            1.00       1.00
##
## Multivariate psrf
##
## 1.03
```

```r
plot(fit)
```

Trace of lambda   **Density of lambda**

N = 5000   Bandwidth = 0.1593

Trace of p   **Density of p**

N = 5000   Bandwidth = 0.01985

Trace of q   **Density of q**

N = 5000   Bandwidth = 0.006274

```
pairs(fit)
```