Abundance models with detection error

Peter Solymos and Subhash Lele

July 16, 2016 — Madison, WI — NACCB Congress

Contents

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

0.0.1 Assumptions

- Replicate surveys,
- independence,
- closed population.

0.0.2 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
```

```
library(rjags)
```

```
## Linked to JAGS 4.2.0
```

```
## 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
```

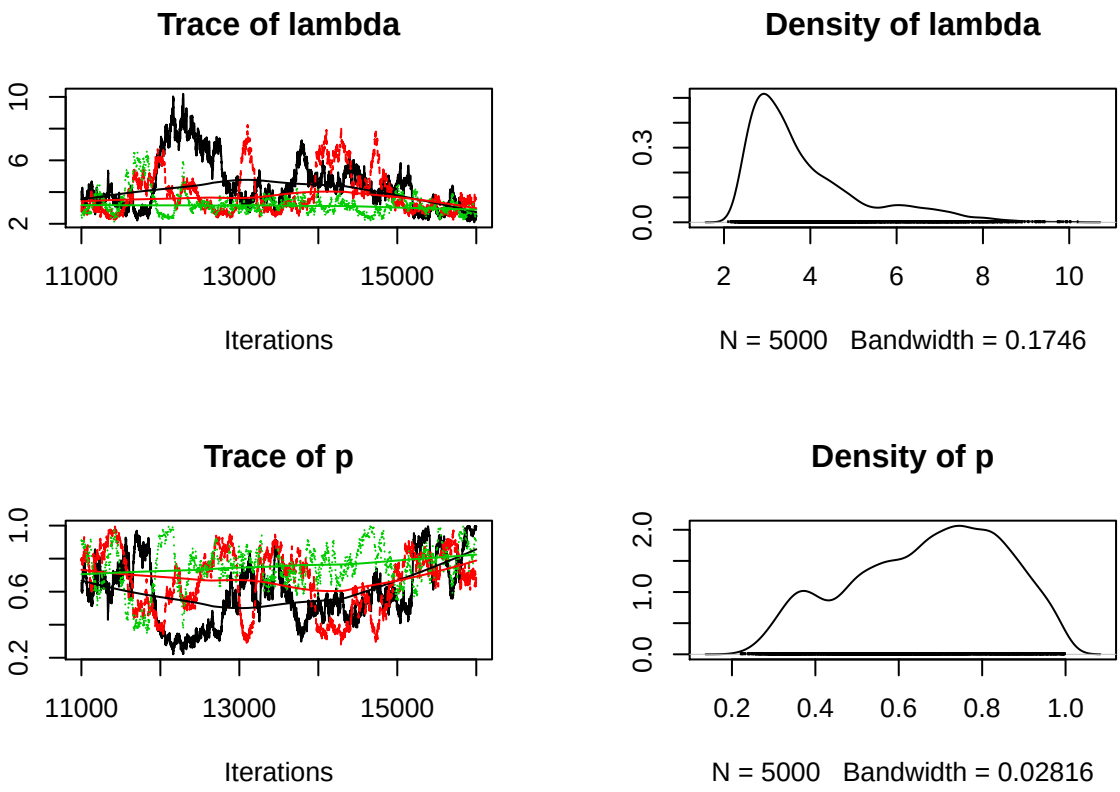
```r
summary(fit)
```

```
##
## 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 3.8772 1.3614 0.011116        0.24371
## p      0.6684 0.1818 0.001484        0.03107
##
## 2. Quantiles for each variable:
##
##         2.5%    25%    50%    75%  97.5%
## lambda 2.410 2.9060 3.4212 4.4159 7.4313
## p      0.316 0.5345 0.6899 0.8133 0.9646
```

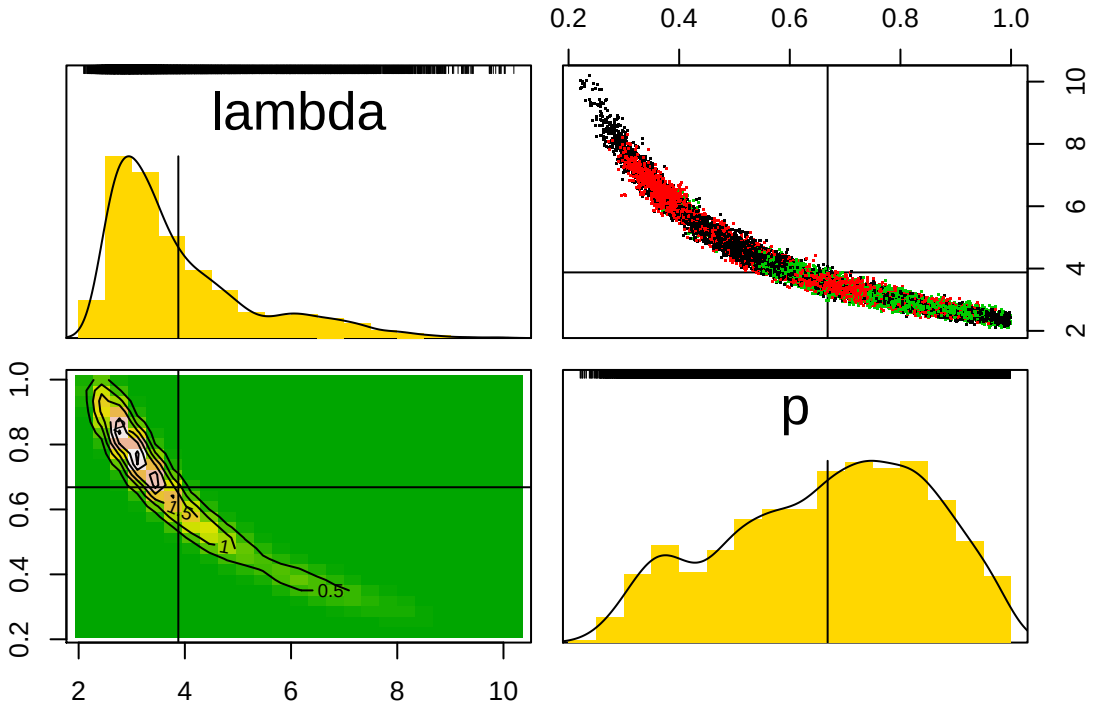
```r
gelman.diag(fit)
```

```
## Potential scale reduction factors:
##
##        Point est. Upper C.I.
## lambda       1.20       1.62
## p            1.17       1.50
##
## Multivariate psrf
##
## 1.16
```

```r
plot(fit)
```


Trace of lambda
2
6
10
11000
13000
15000
Iterations
Density of lambda
0.0
0.3
2
4
6
8
10
N = 5000 Bandwidth = 0.1746
Trace of p
0.2
0.6
1.0
11000
13000
15000
Iterations
Density of p
0.0
1.0
2.0
0.2
0.4
0.6
0.8
1.0
N = 5000 Bandwidth = 0.02816

```
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
##
```

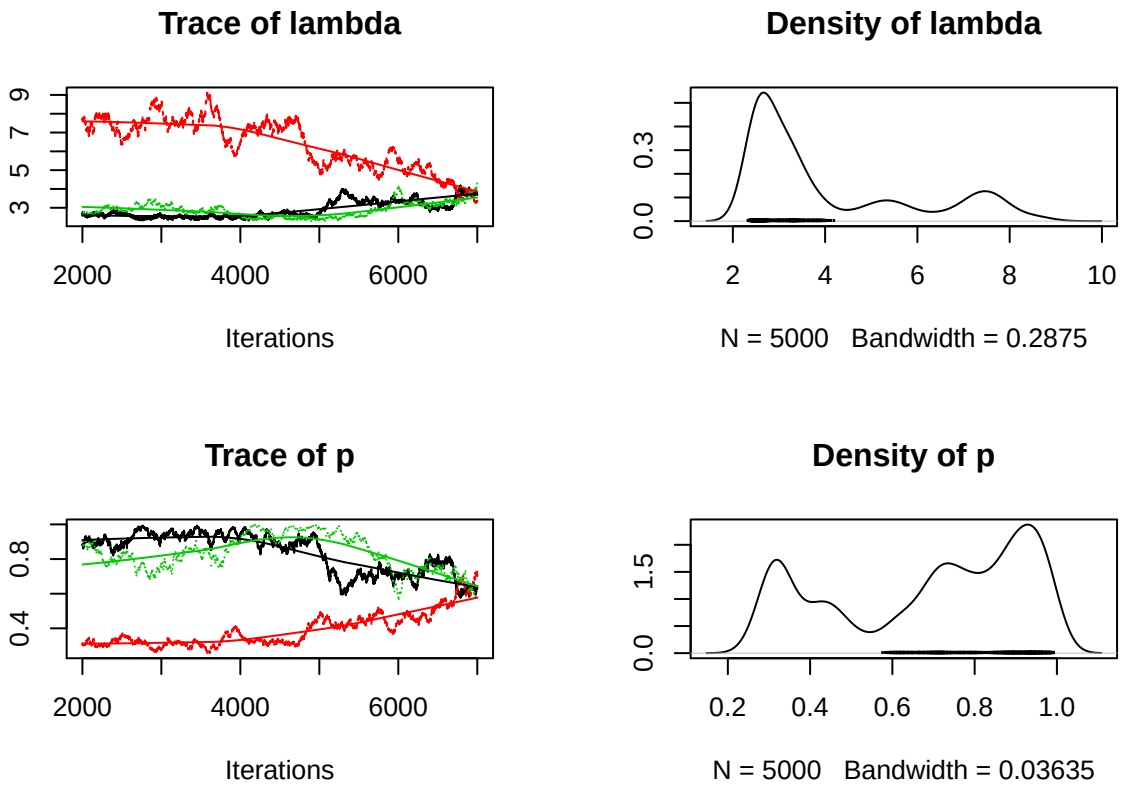
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 400
##    Unobserved stochastic nodes: 402
##    Total graph size: 808
##
## Initializing model
##
##
## Fitting model with 4 clones
##
## Compiling model graph
##    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 4.049 1.8562 5.2501 0.015156        0.23471 3.185
```

```
## p      0.686 0.2346 0.6636 0.001916        0.02844 3.210
##
## 2. Quantiles for each variable:
##
##          2.5%    25%    50%    75%  97.5%
## lambda 2.3934 2.6338 3.1912 5.2669 8.0337
## p      0.2934 0.4484 0.7395 0.8969 0.9829
```

```
plot(dcfit)
```

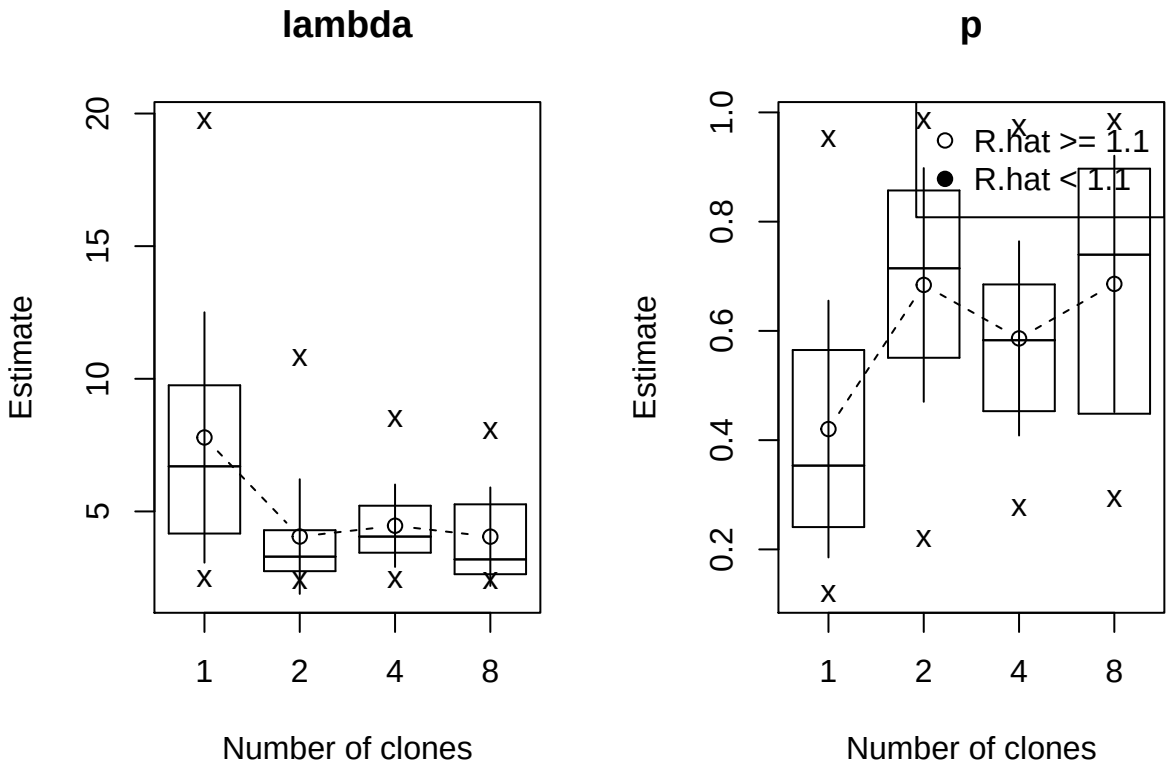



```
dctable(dcfit)
```

```
## $lambda
##   n.clones     mean       sd     2.5%      25%      50%      75%     97.5%
## 1        1 7.787314 4.718931 2.468513 4.167002 6.699694 9.754983 19.718905
## 2        2 4.052545 2.160564 2.373586 2.748180 3.294568 4.291971 10.803523
## 3        4 4.460779 1.553142 2.435699 3.443647 4.051679 5.215005  8.505250
## 4        8 4.048604 1.856179 2.393372 2.633757 3.191189 5.266904  8.033707
##      r.hat
## 1 1.265619
## 2 1.882788
## 3 1.275288
## 4 3.185272
##
## $p
```

```
##   n.clones      mean        sd      2.5%       25%       50%       75%
## 1        1 0.4203190 0.2349991 0.1185850 0.2408375 0.3533848 0.5649739
## 2        2 0.6841239 0.2138829 0.2188158 0.5507603 0.7145696 0.8570387
## 3        4 0.5862330 0.1777523 0.2770057 0.4530742 0.5828814 0.6850404
## 4        8 0.6860135 0.2346274 0.2933780 0.4483676 0.7394817 0.8969433
##       97.5%    r.hat
## 1 0.9514862 1.438727
## 2 0.9842449 1.868877
## 3 0.9728823 1.324340
## 4 0.9828865 3.209791
##
## attr(,"class")
## [1] "dctable"
```

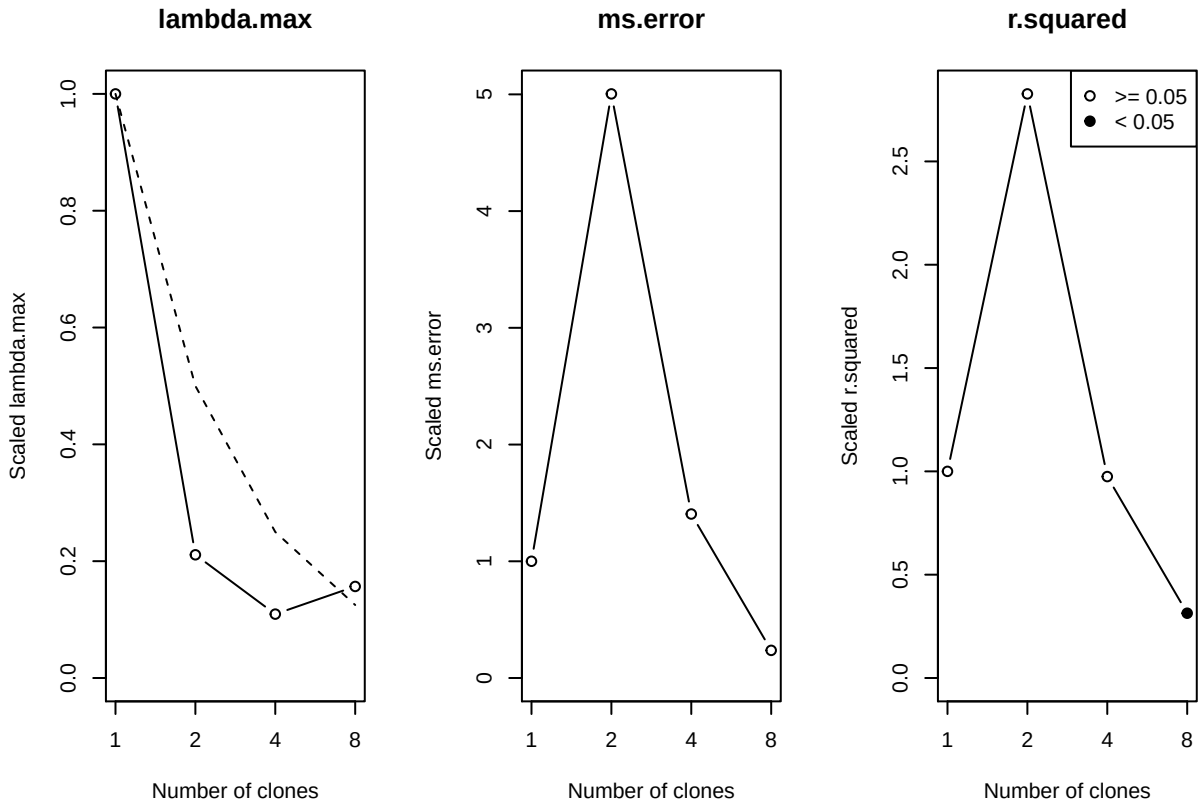
```
plot(dctable(dcfit))
```



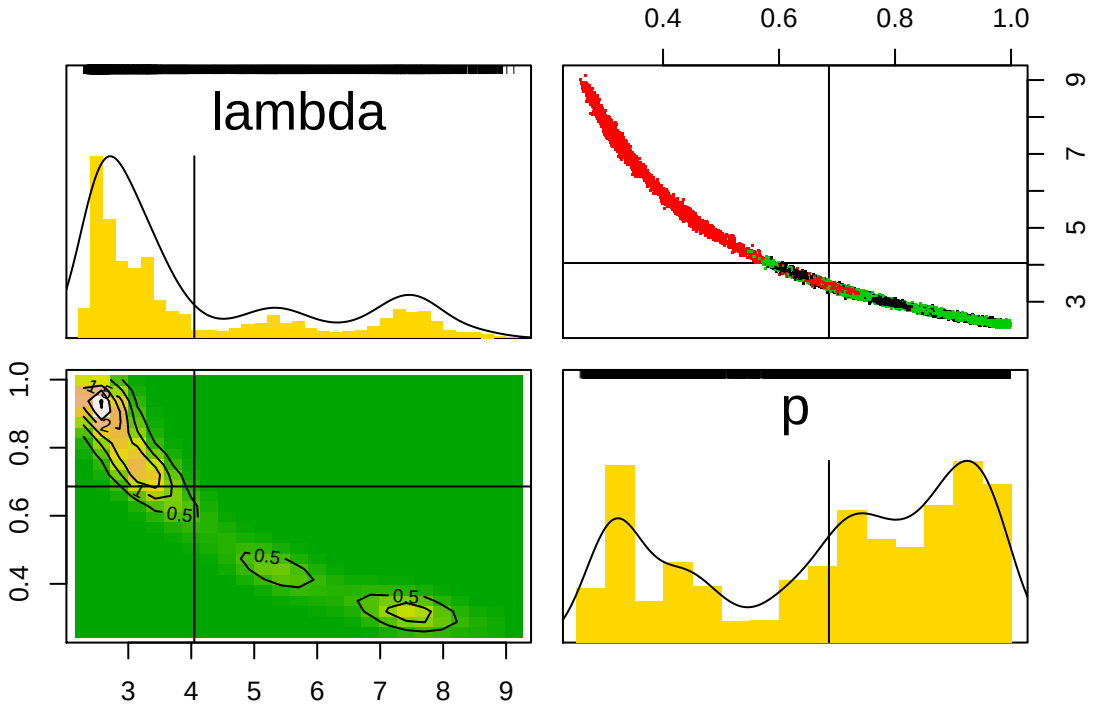

```
dcdiag(dcfit)
```

```
##   n.clones lambda.max  ms.error  r.squared    r.hat
## 1        1  22.305793 0.8343778 0.08090730 1.345845
## 2        2   4.704787 4.1753983 0.22870954 1.668033
## 3        4   2.439339 1.1722108 0.07886798 1.267062
## 4        8   3.496027 0.1975575 0.02536581 2.822644
```

```
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)
```

```
## Loading required package: Formula

## Loading required package: stats4

## Loading required package: pbapply

## 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
```

0.1 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)
}
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
```

0.1.1 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)
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
```

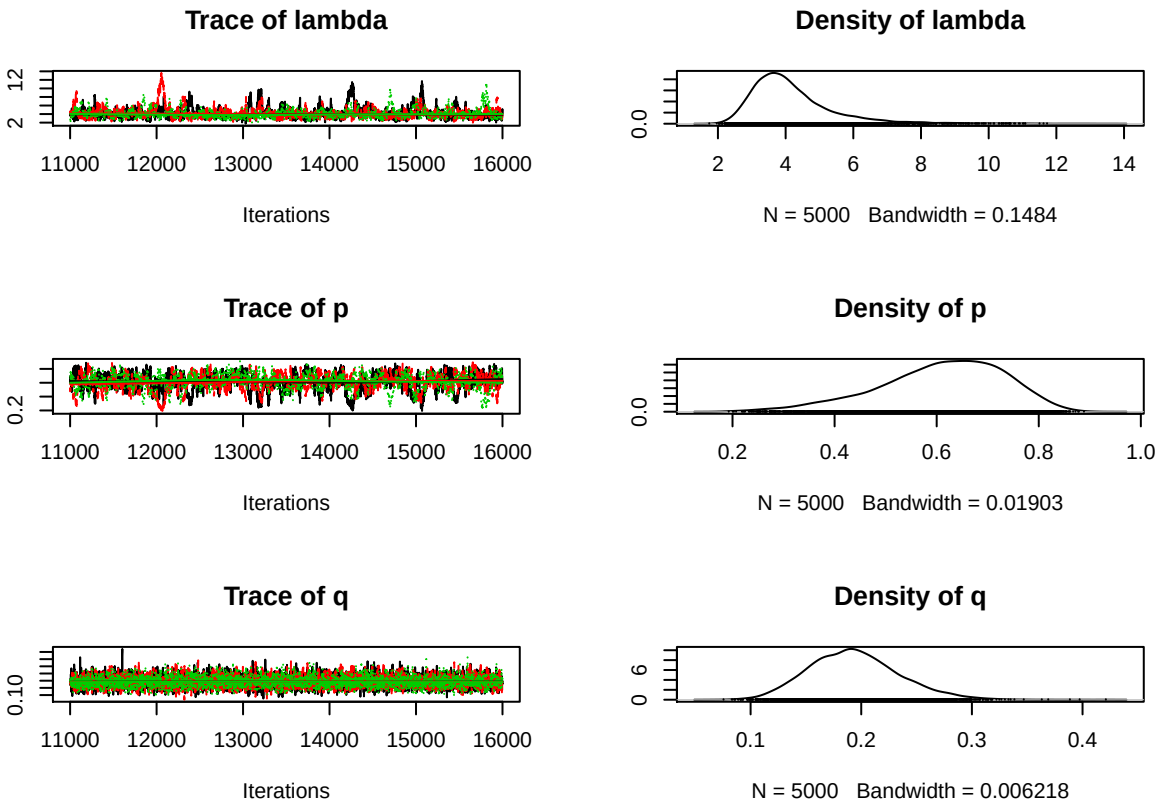
```
summary(fit)
```

```
##
## 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 4.1824 1.27153 0.0103820      0.0745688
## p      0.6091 0.12283 0.0010029      0.0065324
## q      0.1937 0.04038 0.0003297      0.0004658
##
## 2. Quantiles for each variable:
##
##          2.5%    25%    50%    75%  97.5%
## lambda 2.5778 3.3597 3.9086 4.6438 7.5547
## p      0.3325 0.5343 0.6223 0.7003 0.8079
## q      0.1214 0.1652 0.1915 0.2190 0.2802
```

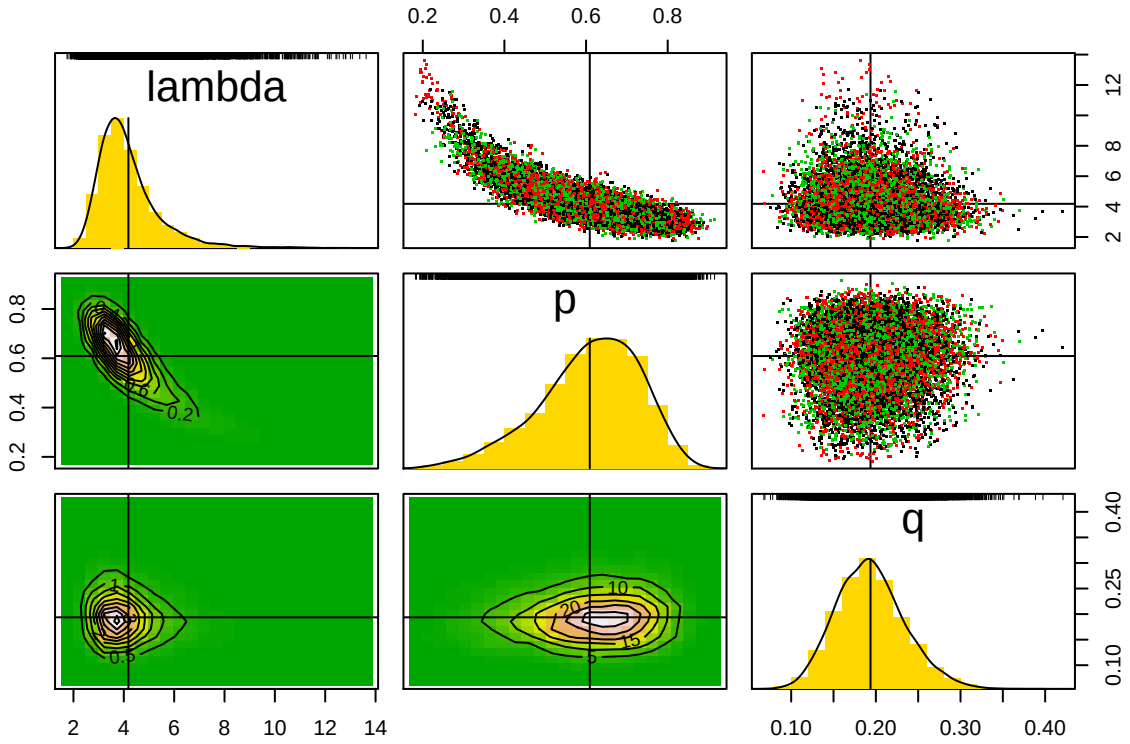
```
gelman.diag(fit)
```

```
## 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)
```


Trace of lambda
2 12
11000 12000 13000 14000 15000 16000
Iterations
Density of lambda
0.0
2 4 6 8 10 12 14
N = 5000 Bandwidth = 0.1484
Trace of p
0.2
11000 12000 13000 14000 15000 16000
Iterations
Density of p
0.0
0.2 0.4 0.6 0.8 1.0
N = 5000 Bandwidth = 0.01903
Trace of q
0.10
11000 12000 13000 14000 15000 16000
Iterations
Density of q
0 6
0.1 0.2 0.3 0.4
N = 5000 Bandwidth = 0.006218

```
pairs(fit)
```


lambda
p
q
0.2
0.4
0.6
0.8
2
4
6
8
10
12
14
0.10
0.20
0.30
0.40
0.25